



# AFP Toolbox for Multiple Operating Systems User's Guide

*Version 1 Release 1*





# AFP Toolbox for Multiple Operating Systems User's Guide

*Version 1 Release 1*

**Note**

Before using this information and the product it supports, be sure to read the general information in Notices on page 472.

**Fifth Edition (February 2002)**

This edition applies to Version 1 Release 1.0 of the AFP™ Toolbox for Multiple Operating Systems, Program Numbers 5765-594, 5655-A25, 5798-AF2, and 5798-AF4, and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Be sure to use the correct edition for the level of the product.

Order publications through your IBM® representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

The IBM Printing Systems Division welcomes your comments. A form for reader's comments is provided at the back of this publication. If the form has been removed, you may send your comments to the following address:

INFORMATION DEVELOPMENT THE IBM PRINTING SYSTEMS DIVISION DEPARTMENT H7FE BUILDING 004M  
PO BOX 1900 BOULDER CO 80301-9191 U.S.A.

If you prefer to send comments electronically, use one of the following methods:

- Internet: [printpub@us.ibm.com](mailto:printpub@us.ibm.com)
- Fax: 1-800-524-1519 or 1-303-924-6873

**Internet**

Visit our home page at <http://www.ibm.com/printers>

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1995, 2002. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Figures</b> . . . . .	ix
<b>About This Publication</b> . . . . .	xi
Who Should Read This Publication . . . . .	xi
How This Publication Is Organized . . . . .	xi
Related Information . . . . .	xi
<b>I Summary of Changes</b> . . . . .	xiii
<b>Part 1. What is AFP Toolbox and How Does it Work?</b> . . . . .	1
<b>Chapter 1. Overview</b> . . . . .	3
The AFP Toolbox C++ Object Library . . . . .	5
The AFP Toolbox C Library . . . . .	6
The AFP Toolbox COBOL Interface . . . . .	6
The AFP Toolbox RPG Interface . . . . .	6
The FontMeister C Library. . . . .	6
What is Advanced Function Presentation? . . . . .	6
AFP Documents, Pages, and Resources . . . . .	6
AFP Documents and Pages . . . . .	7
AFP Data and Resource Objects . . . . .	9
AFP Data Objects . . . . .	10
AFP Resource Objects . . . . .	10
Grouping, Archiving, and Linking Functions . . . . .	12
The Grouping Function . . . . .	12
The Linking Function . . . . .	12
<b>Chapter 2. Using AFP Toolbox</b> . . . . .	13
C Sample Notes . . . . .	15
C++ Sample Notes . . . . .	15
COBOL Sample Notes . . . . .	15
Getting Started . . . . .	15
Beginning the Session . . . . .	16
Defining Fonts . . . . .	17
Beginning a Document . . . . .	18
Indexing . . . . .	19
Creating Page Objects . . . . .	20
Beginning a Page . . . . .	21
Putting Data on the Page . . . . .	22
Determining Page Breaks . . . . .	28
Ending a Page . . . . .	29
Ending a Document . . . . .	31
Ending the Session. . . . .	32
Working with Bar Codes . . . . .	33
Begin a Bar Code . . . . .	33
Specifying Bar Code Data . . . . .	34
Place Bar Code . . . . .	35
Working with Tables . . . . .	36
Example . . . . .	37
More About Tables . . . . .	42
Step 1. Sketch the Row . . . . .	42
Step 2. Define all of the Fields in the Row . . . . .	42
Step 3. Form a Grid . . . . .	42

Step 4. Determine the Number of Columns and Their Widths . . . . .	43
Step 5. Determine the Number Subrows and Their Depths . . . . .	43
Step 6. Determine the Arrangement of Each Field Within Each Subrow . . . . .	43
Step 7. Define The Row . . . . .	43
Working with Code Pages . . . . .	44
Using Fonts with AFP Toolbox . . . . .	46
Printing and Viewing the Output File . . . . .	47

---

## **Part 2. AFP Toolbox Language Reference . . . . . 49**

### **Chapter 3. AFP Toolbox Language Reference . . . . . 53**

Using the AFP Toolbox Calls to Build an AFP Document . . . . .	53
Hierarchy of Calls . . . . .	53
Session-level calls . . . . .	54
Document-level Calls . . . . .	54
Page-level Calls . . . . .	55
Return Codes . . . . .	58
Understanding Handles . . . . .	59
Providing Positioning and Size Information . . . . .	59
Format of the Function Call Descriptions . . . . .	60
Including Header Files. . . . .	61
Bar Code Data . . . . .	62
Begin Bar Code . . . . .	66
Begin Box . . . . .	73
Begin Color . . . . .	75
Begin Document . . . . .	78
Begin Field . . . . .	82
Begin Group . . . . .	84
Begin Image . . . . .	86
Begin Page . . . . .	91
Begin Paragraph . . . . .	93
Begin Row . . . . .	96
Begin Rule . . . . .	97
Begin Session . . . . .	100
Begin Shade . . . . .	101
Begin Table . . . . .	103
Begin Underscore . . . . .	106
Create Link . . . . .	108
I Data Matrix Bar Code Data . . . . .	111
Define Double-Byte Font By Attribute . . . . .	114
Define Field . . . . .	117
Define Font By Attribute . . . . .	120
Define Font By Attributes With Scaling . . . . .	123
Define Font By Name . . . . .	126
Define Row . . . . .	128
Delete Font . . . . .	131
Delete Page . . . . .	132
End Bar Code . . . . .	134
End Box . . . . .	135
End Color . . . . .	136
End Document . . . . .	137
End Field . . . . .	138
End Group . . . . .	139
End Image . . . . .	140
End Page . . . . .	141
End Paragraph . . . . .	142

End Row . . . . .	143
End Rule . . . . .	145
End Session . . . . .	146
End Shade . . . . .	147
End Table . . . . .	148
End Underscore . . . . .	149
Get Buffer . . . . .	150
Graphic Inline . . . . .	152
Horizontal Move . . . . .	154
Horizontal Move To . . . . .	156
Image Data . . . . .	158
Image Inline . . . . .	160
Include Object . . . . .	162
Include Overlay . . . . .	167
Include Page Segment . . . . .	169
Invoke Medium Map . . . . .	171
Keep Object . . . . .	173
I MaxiCode Bar Code Data . . . . .	175
Measure Double-Byte String . . . . .	178
Measure String . . . . .	180
Next Line . . . . .	183
Output Comment . . . . .	184
I PDF417 Bar Code Data . . . . .	186
Preload Object . . . . .	189
Preload Overlay . . . . .	191
Put Box . . . . .	192
Put Color . . . . .	194
Put Double-Byte Text . . . . .	196
Put Horizontal Rule . . . . .	198
Put Shade . . . . .	200
Put Tag . . . . .	201
Put Text . . . . .	203
Put Text in Table Field . . . . .	205
Put Vertical Rule . . . . .	207
Query . . . . .	209
Repeat String . . . . .	211
Rotate Overlay . . . . .	213
Set Color . . . . .	215
Set Data Stream Codepage . . . . .	217
Set Extended Color . . . . .	219
Set Font . . . . .	221
Set Font Type . . . . .	223
Set Input Codepage . . . . .	225
Set Input Double-Byte Codeset . . . . .	228
Set Media Size . . . . .	230
Set Object Color Profile . . . . .	232
Set Position . . . . .	234
Set Text Codepage . . . . .	236
Set Text Double-Byte Codeset . . . . .	238
Set Text Orientation . . . . .	240
Set Units . . . . .	242
Translate And Measure Double-Byte String . . . . .	244
Translate And Measure String . . . . .	246
Vertical Move . . . . .	249
Vertical Move To . . . . .	251
Write Double-Byte String . . . . .	253

Write String . . . . .	256
Font Return Codes . . . . .	259

<b>Chapter 4. AFP Toolbox GOCA Functions . . . . .</b>	<b>261</b>
Graphic Presentation Space Concepts . . . . .	261
C++ Objects . . . . .	261
C and COBOL Function Descriptions . . . . .	261
Header files and Copy Books . . . . .	262
Begin Graphic Object . . . . .	263
End Graphic Object . . . . .	266
GOCA Begin Area . . . . .	267
GOCA Character String . . . . .	269
GOCA Circle . . . . .	271
GOCA Draw Box . . . . .	272
GOCA Draw Objects . . . . .	274
GOCA Ellipse . . . . .	277
GOCA End Area . . . . .	280
GOCA No Operation . . . . .	281
GOCA Partial Circle . . . . .	282
GOCA Partial Ellipse . . . . .	284
GOCA Query GPS Position . . . . .	286
GOCA Set Character Parameters . . . . .	287
GOCA Set Font . . . . .	290
GOCA Set GPS Position . . . . .	291
GOCA Set Extended Color . . . . .	292
GOCA Set Line Parameters . . . . .	294
GOCA Set Marker Parameters . . . . .	296
GOCA Set Pattern Symbol . . . . .	298
GOCA Set Process Color . . . . .	301

---

## **Part 3. AFP Toolbox for C++ Language . . . . . 305**

<b>Chapter 5. AFP Toolbox C++ Object Library . . . . .</b>	<b>309</b>
Using Objects to Build an AFP Document . . . . .	309
AEStrng . . . . .	310
Including C++ Header Files . . . . .	311
Document . . . . .	312
Declarations for the Document Class . . . . .	312
Constructors and Operators . . . . .	312
Public Members of Document . . . . .	312
AFPDoc . . . . .	314
Declarations for the AFPDoc Class . . . . .	314
SetDocumentCodepage . . . . .	314
SetInputCodepage . . . . .	314
SetInputDbCodeset . . . . .	315
Constructors and Operators . . . . .	315
Public Members of AFPDoc . . . . .	315
Page Class . . . . .	318
Declarations for the Page Class . . . . .	318
Constructors and Operators . . . . .	318
Public Members of Page . . . . .	318
AFPPage . . . . .	319
Declarations for the AFPPage Class . . . . .	319
Constructors and Operators . . . . .	320
Public Members of AFPPage . . . . .	320
PText . . . . .	326



Declarations for the PText Class . . . . .	326
Constructors and Operators . . . . .	327
Public Members of PText . . . . .	327
OCA . . . . .	333
Declarations for the OCA Class . . . . .	333
Constructors and Operators . . . . .	334
Public Members of OCA . . . . .	334
Image Object Content Architecture (IOCA) . . . . .	337
Declarations for the IOCA Class . . . . .	337
Constructors and Operators . . . . .	338
Public Members of IOCA . . . . .	339
Bar Code Object Content Architecture (BCOCA) . . . . .	342
Declarations for the BCOCA Class . . . . .	342
Constructors and Operators . . . . .	342
Public Members of BCOCA . . . . .	344
Graphic Object Content Architecture (GOCA) . . . . .	346
Declarations for the GOCA Class . . . . .	346
Constructors and Operators . . . . .	347
Public Members of GOCA . . . . .	347
 <b>Chapter 6. Defining Fonts with C++ Language Programs</b> . . . . .	 355
Define Double-Byte Font By Attribute . . . . .	356
Define Font By Attribute . . . . .	358
Define Font By Attribute With Scaling . . . . .	360
Define Font By Name . . . . .	362
Measure Double-Byte String . . . . .	363
Measure String . . . . .	365
Set Font Type . . . . .	367
Translate And Measure Double-Byte String . . . . .	368
Translate And Measure String . . . . .	370
 <b>Part 4. Appendixes</b> . . . . .	 373
 <b>Appendix A. AFP Toolbox and the OS/390 Environment</b> . . . . .	 375
Prerequisites and System Requirements . . . . .	375
Using Fonts with the AFP Toolbox . . . . .	375
Compiling and Running the Sample Programs . . . . .	376
COBOL Sample . . . . .	376
C Sample . . . . .	378
C++ Sample . . . . .	380
CBLGetBuffer Program . . . . .	382
Running the MVS Toolbox on an Open Edition System . . . . .	385
OS/390 Unix System Services Environment Variables . . . . .	385
Sending Output Directly to MVS SPOOL . . . . .	386
Troubleshooting . . . . .	387
 <b>Appendix B. AFP Toolbox and the AIX Environment</b> . . . . .	 389
The FontMeister Directory . . . . .	389
Compiling and Running the Sample Programs . . . . .	389
Viewing the Online Documentation . . . . .	389
Useful Tools . . . . .	390
AFPDMP . . . . .	390
FLIPLIST . . . . .	390
Updating Your Font Index . . . . .	390
Troubleshooting . . . . .	393

<b>Appendix C. AFP Toolbox and the AS/400 Environment.</b>	395
Prerequisites and System Requirements	395
Using Fonts with the AFP Toolbox	395
Compiling and Running the Sample Programs	395
COBOL Sample	396
C Sample	396
RPG Sample.	397
I Updating Your Font Index	397
Troubleshooting	399
 <b>Appendix D. Migrating Applications from AFP API to AFP Toolbox</b>	401
Differences in Codepage Processing	403
Functions Provided in Toolbox That Are Not in API.	404
 <b>Appendix E. AFP Toolbox Table Samples</b>	405
Sample in C	405
Sample in C++	450
Sample in COBOL	462
 <b>Notices</b>	471
Notices.	472
COPYRIGHT LICENSE	473
Programming Interfaces	473
Trademarks	473
EuroReady	474
Year 2000 Ready	474
 <b>Glossary</b>	475
Source Identifiers	475
References	475
 <b>Bibliography</b>	485
Advanced Function Presentation (AFP)	485
BCOCA	485
OS/400 and iSeries	486
MVS.	486
Infoprint Server for OS/390	486
Print Services Facility (PSF) for OS/390.	486
Fonts	487
Text Processing	487
Infoprint Manager	488
Printers.	488
TCP/IP	489
TCP/IP for MVS	489
VTAM and NCP	489
System Network Architecture (SNA)	490
IBM Architecture Publications	490
IBM Advanced Function Presentation Publications	490
 <b>Index</b>	491

---

## Figures

1.	MO:DCA Presentation Document Components . . . . .	4
2.	AFP Toolbox Document and Page Structure . . . . .	5
3.	Logical Page Coordinate System . . . . .	7
4.	Medium Coordinate System . . . . .	8
5.	Relationship between the Logical Page and Media Coordinate Systems . . . . .	9
6.	Sample Document . . . . .	14
7.	Bar Code Presentation Space . . . . .	33
8.	How Code Pages Work . . . . .	45
9.	C syntax . . . . .	111
10.	COBOL syntax . . . . .	111
11.	C syntax . . . . .	175
12.	COBOL syntax . . . . .	175
13.	C syntax . . . . .	186
14.	COBOL syntax . . . . .	186
15.	Text Orientation Examples . . . . .	241
16.	Graphics Presentation Space . . . . .	261
17.	Graphics Presentation Space . . . . .	264
18.	Fillels . . . . .	275
19.	Ellipses . . . . .	278
20.	Partial Circle . . . . .	283
21.	Partial Ellipse . . . . .	285
22.	Character Angles and Directions. . . . .	289
23.	Fill Patterns . . . . .	299
24.	Text Orientation Examples . . . . .	331
25.	Object and Object Content Positioning . . . . .	336
26.	GOCA Presentation Space. . . . .	347
27.	COBOL Sample. . . . .	378



---

## About This Publication

This publication provides information about using AFP Toolbox for multiple operating systems (licensed program number 5765-594) , AFP Toolbox for MVS™ (licensed program number 5655-A25), and the AFP Toolbox component of AFP PrintSuite for AS/400® (licensed program number 5798-AF4).

This publication is written with the assumption that you have experience with application programming and with Advanced Function Presentation™ (AFP) printers. Throughout this publication, the term MVS applies to Z/OS, OS/390®, MVS. The term AS/400 applies to AS/400 and iSeries. OS/400® is the operating system of the AS/400.

---

## Who Should Read This Publication

This publication contains information that application programmers can use to develop AFP Toolbox applications to customize AFP documents.

---

## How This Publication Is Organized

This publication is organized into three parts, each part containing multiple chapters, to help you obtain the information you need about AFP Toolbox:

- "Part 1. What is AFP Toolbox and How Does it Work" contains these chapters:
  - Chapter 1, "Overview" on page 3 describes the AFP Toolbox interfaces in general terms.
  - Chapter 2, "Using AFP Toolbox" on page 13 describes a sample application for using AFP Toolbox showing both C and C++ code.
- "Part 2. AFP Toolbox for C++ Language" contains these chapters:
  - Chapter 5, "AFP Toolbox C++ Object Library" on page 309 describes the C++ programming interface of AFP Toolbox.
  - Chapter 6, "Defining Fonts with C++ Language Programs" on page 355 describes the C++ programming interface for defining fonts with the AFP Toolbox.
- "Part 3. AFP Toolbox Language Reference" contains these chapters:
  - Chapter 3, "AFP Toolbox Language Reference" on page 53 describes the C, COBOL, and RPG programming interface for AFP Toolbox.
  - Chapter 4, "AFP Toolbox GOCA Functions" on page 261 describes the functions provided with AFP Toolbox for building GOCA objects and structures.
- "Part 4. Appendixes" contains these appendixes:
  - Appendix A, "AFP Toolbox and the OS/390 Environment" on page 375 describes the MVS programming environment.
  - Appendix B, "AFP Toolbox and the AIX Environment" on page 389 describes the AIX® programming environment.
  - Appendix C, "AFP Toolbox and the AS/400 Environment" on page 395 describes the OS/400 programming environment.
  - Appendix D, "Migrating Applications from AFP API to AFP Toolbox" on page 401 describes the migrating Applications from AFP API to AFP Toolbox.
  - Appendix E, "AFP Toolbox Table Samples" on page 405 gives complete samples for creating tables in AFP Toolbox.

Pertinent Notices, a Glossary, a Bibliography, and an Index are included at the back of the publication.

---

## Related Information

Publications that are referred to in this document or that contain additional information about related products and systems are listed in the Bibliography.

To obtain the latest documentation updates for AFP Toolbox, refer to the AFP Toolbox new functions and enhancements Web page:

<http://www.ibm.com/printers/R5PSC.NSF/Web/tbxupdt>

For online technical support for this product, refer to the Software support Web page:

<http://www.ibm.com/software/support/>

For online technical support for the RS/6000®, iSeries, AS/400, zSeries, or OS/390, refer to the Software support Web page:

<http://www.ibm.com/servers/support/>

---

## Summary of Changes

### Summary of Changes for IBM AFP Toolbox for Multiple Operating Systems User's Guide, S544-5292-04

This publication contains additions and changes to information previously presented in *IBM AFP Toolbox for Multiple Operating Systems User's Guide*, S544-5625-03. The technical additions and changes are marked with a revision bar ( | ) in the left margin.

The following information is new or updated:

- Chapter 2, "Using AFP Toolbox" on page 13 has the following changes:
  - A pointer to the new two-dimensional bar codes has been added to "Specifying Bar Code Data" on page 34
- Chapter 3, "AFP Toolbox Language Reference" on page 53 has the following changes:
  - Two-dimensional bar code information has been added to "Bar Code Data" on page 62
  - The following calls have been added:
    - "Data Matrix Bar Code Data" on page 111
    - "MaxiCode Bar Code Data" on page 175
    - "PDF417 Bar Code Data" on page 186





---

## Part 1. What is AFP Toolbox and How Does it Work?

<b>Chapter 1. Overview</b>	3
The AFP Toolbox C++ Object Library	5
The AFP Toolbox C Library	6
The AFP Toolbox COBOL Interface	6
The AFP Toolbox RPG Interface	6
The FontMeister C Library	6
What is Advanced Function Presentation?	6
AFP Documents, Pages, and Resources	6
AFP Documents and Pages	7
AFP Data and Resource Objects	9
AFP Data Objects	10
AFP Resource Objects	10
Fonts	10
Page Segments	11
Overlays	11
Form Definitions	11
Page Definitions	11
Grouping, Archiving, and Linking Functions	12
The Grouping Function	12
The Linking Function	12
<b>Chapter 2. Using AFP Toolbox</b>	13
C Sample Notes	15
C++ Sample Notes	15
COBOL Sample Notes	15
Getting Started	15
Beginning the Session	16
Defining Fonts	17
Beginning a Document	18
Indexing	19
Creating Page Objects	20
Beginning a Page	21
Putting Data on the Page	22
Strings of Text	22
Rules	24
Boxes	24
Resources	25
Paragraphs	26
Determining Page Breaks	28
Ending a Page	29
Ending a Document	31
Ending the Session	32
Working with Bar Codes	33
Begin a Bar Code	33
Specifying Bar Code Data	34
Place Bar Code	35
Working with Tables	36
Example	37
Define Field	37
Define Row	38
Begin Table	39
Begin Row	39
Begin Field	39

Set Font . . . . .	39
Trim . . . . .	40
Put Text . . . . .	40
End Field . . . . .	40
Begin Field . . . . .	40
Put Text . . . . .	40
End Field . . . . .	41
Begin Field . . . . .	41
Put Text . . . . .	41
End Field . . . . .	41
End Row . . . . .	41
End Table . . . . .	42
More About Tables . . . . .	42
Step 1. Sketch the Row . . . . .	42
Step 2. Define all of the Fields in the Row . . . . .	42
Step 3. Form a Grid . . . . .	42
Step 4. Determine the Number of Columns and Their Widths . . . . .	43
Step 5. Determine the Number Subrows and Their Depths . . . . .	43
Step 6. Determine the Arrangement of Each Field Within Each Subrow . . . . .	43
Step 7. Define The Row . . . . .	43
Working with Code Pages . . . . .	44
Using Fonts with AFP Toolbox . . . . .	46
Printing and Viewing the Output File . . . . .	47

---

## Chapter 1. Overview

The IBM AFP Toolbox helps application programmers produce compound documents using the Mixed Object: Document Content Architecture (MO:DCA) data stream.<sup>1</sup> In AFP terms, a compound document is a collection of data objects that makes up the document's content, the associated resources, and formatting specifications that dictate the processing functions to be performed on the document's content. A MO:DCA document can include a mixture of presentation text, image, graphics, and bar code data objects. Figure 1 on page 4 shows a sample MO:DCA compound document. For a detailed description of the MO:DCA data stream, see the *MO:DCA Reference*.

---

1. Some people use the terms MO:DCA and AFP Data Stream (AFPDs) interchangeably although they are not really the same thing. In this document, we refer to the MO:DCA data stream.

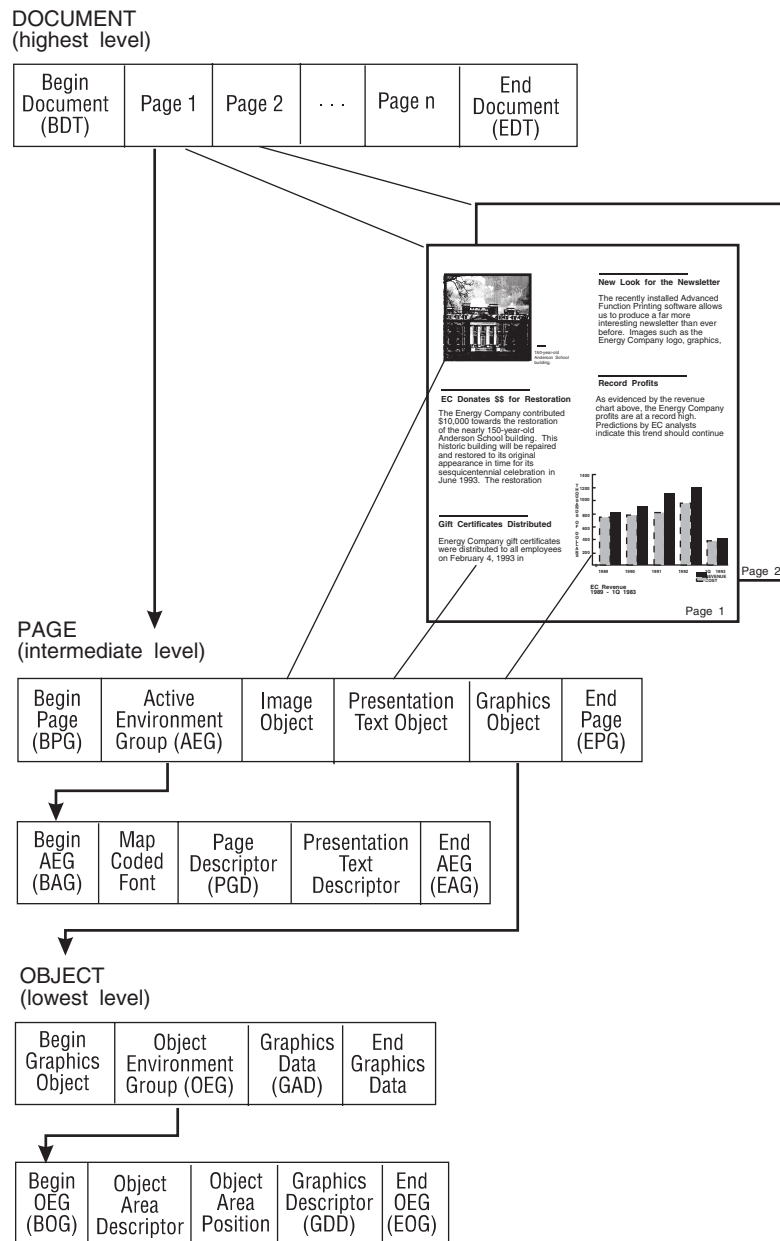


Figure 1. MO:DCA Presentation Document Components

The AFP Toolbox is designed to hide the details of the data stream from users that need to generate it.

Applications that use the AFP Toolbox library functions need to follow the AFP hierarchy when building compound documents: documents made up of pages and the pages made up of objects, as shown in Figure 2 on page 5.

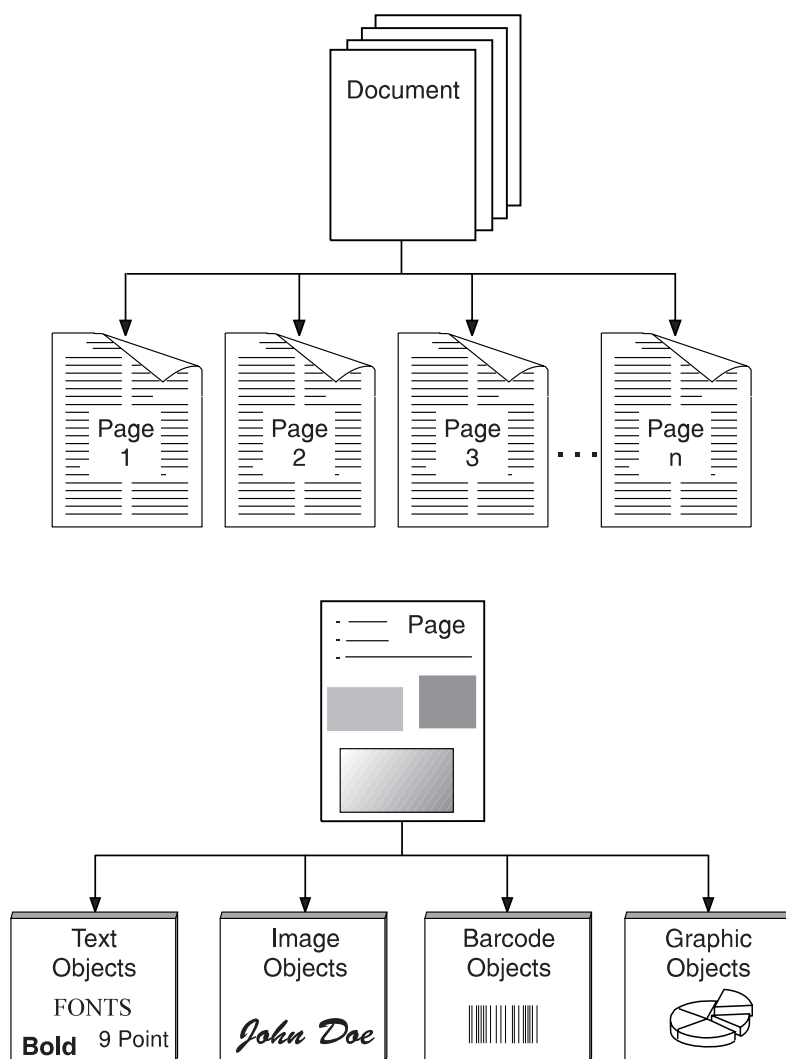


Figure 2. AFP Toolbox Document and Page Structure

Chapter 5, “AFP Toolbox C++ Object Library” on page 309 and Chapter 3, “AFP Toolbox Language Reference” on page 53 describe the supported objects and show the syntax (for C++, C, COBOL, and RPG) of the function calls used to generate documents and pages containing these objects.

## The AFP Toolbox C++ Object Library

**Note:** The C/C++ Compiler is not available in OS/400 versions prior to V5R1.

The majority of Toolbox C++ objects are unpublished and used mostly for internal processing within the Toolbox itself. Therefore, there is only limited support on a variety of operations on these objects or pointers to these objects such as: initialization, de-referencing, I/O, assignments, comparisons, and arithmetic.

In order to use the structured-field objects, the caller must understand the structure of the AFP documents and pages (that is, the order and hierarchy of the structured fields.) The structured field objects are not documented as an external programming interface in the first release of this product. Most programmers will prefer using the high level object interface over the structured field interface. For more information about the structured field interface, contact your IBM Sales or Service Representative.

The high-level objects in the AFP Toolbox C++ Object Library are modeled around the components of the MO:DCA data stream and most AFP documents. AFP Toolbox C++ Object Library users must understand the general or high-level structure of an AFP document, and be aware that AFP data is being produced. With the functions provided for each object type (document, page, text, and image), application developers can construct applications without requiring specific knowledge of the MO:DCA data stream syntax or semantics. Therefore this interface is designed to hide both the semantics and the syntax of the MO:DCA data stream from an application.

---

## The AFP Toolbox C Library

The AFP Toolbox C Library is a C language interface to the high-level AFP Toolbox C++ Object Library objects. Using procedural function calls, programmers can access the functions of the C++ objects in a manner consistent with C language procedural programming while retaining the function provided in the object interface. The C interface is supported on all platforms on which Toolbox runs.

---

## The AFP Toolbox COBOL Interface

The AFP Toolbox COBOL Interface is a COBOL language interface provided in MVS and OS/400 to the high-level AFP Toolbox C++ Object Library objects. Using procedural function calls, programmers can access the functions of the C++ objects in a manner consistent with COBOL language procedural programming while retaining the function provided in the object interface.

---

## The AFP Toolbox RPG Interface

The AFP Toolbox RPG Interface is a RPG language interface provided in OS/400 only to the high-level AFP Toolbox C++ Object Library objects. Using procedural function calls, programmers can access the functions of the C++ objects in a manner consistent with RPG language procedural programming while retaining the function provided in the object interface.

---

## The FontMeister C Library

The FontMeister C Library is a small set of C language functions designed to perform font activities. The functions in the FontMeister interface let you define a font in terms of its attributes (such as 10 point Times New Roman Bold), define a font in terms of a specific set of characters and code page, and measure the width of a text string in a particular font. All of the supported languages use FontMeister to manage the fonts used in a document, and to perform functions such as aligning text.

---

## What is Advanced Function Presentation?

Advanced Function Presentation (AFP) is a collection of hardware and software products that generates high-quality presentation output from data-processing systems to ensure the computer output is more readable and attractive. AFP is implemented with an architected data stream (the MO:DCA data stream) consisting of *structured fields* of presentation information. Structured fields are self-identifying, variable-length records containing control information data. The two major elements in the MO:DCA data stream are the document and several types of resources.

---

## AFP Documents, Pages, and Resources

An AFP document consists of data that has been formatted into a series of one or more MO:DCA data stream pages. Each page contains the actual output text characters for that page, as well as information about the placement of each character. In addition to text, the following types of data can be included on a page:

- Graphics and images (line art, business graphics, and scanned data such as photographs or facsimile)
- Logos
- Signatures
- Lines (rules) and boxes

- Bar code data
- Indexing information

**Note:** This is ignored when the AFP document is printed, but is useful for navigating through a displayed document.

Programs can store these types of data outside the document and can use the types of data in multiple pages within a document or in multiple applications. These external files are called AFP resources.

---

## AFP Documents and Pages

An AFP document consists of one or more pages of AFP data. The AFP page is referred to as a *logical page* since it is not bound to a particular physical entity such as a set form or display. The logical page is the electronic representation of the page that is ultimately printed or viewed.

The logical page has dimensions and a coordinate system similar to that of a physical form or medium. This logical page coordinate system is referred to as the Xp, Yp coordinate system in the MO:DCA data stream and is shown in Figure 3.

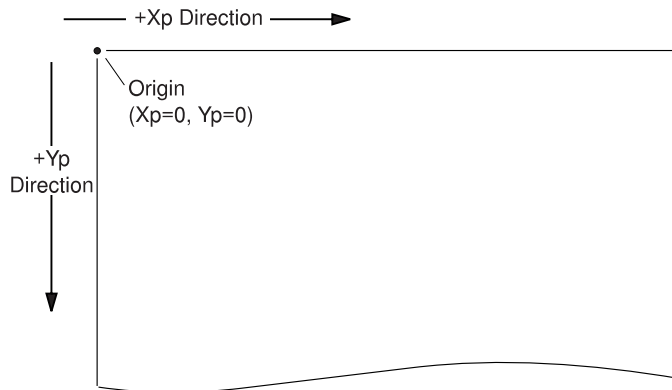


Figure 3. Logical Page Coordinate System

The origin of this system ( $X_p=0$ ,  $Y_p=0$ ) is always defined at the top-left corner of the logical page at the logical page origin. Positive  $X_p$  values begin at the origin and increase along the top of the logical page from left to right. Positive  $Y_p$  values begin at the origin and increase along the left side of the logical page from top to bottom. The size and units of measurement for each logical page of AFP data are specified within the page itself.

The physical medium, whether it is a form or display, also has dimensions and a coordinate system. This medium coordinate system is referred to as the  $X_m$ ,  $Y_m$  coordinate system in the MO:DCA data stream and is shown in Figure 4 on page 8.

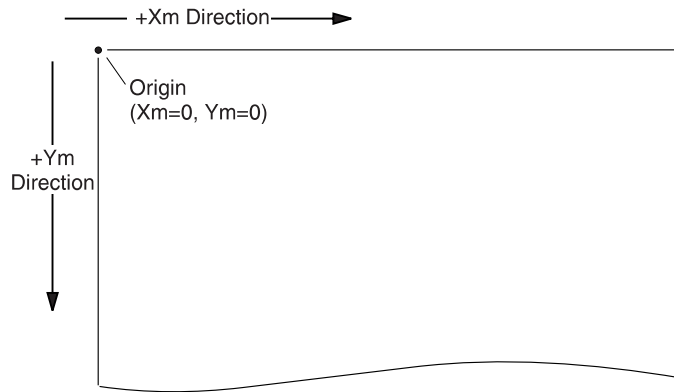


Figure 4. Medium Coordinate System

The origin of this system ( $X_m=0$ ,  $Y_m=0$ ) is always defined at the top-left corner of the medium at the media origin. Positive  $X_m$  values begin at the origin and increase along the top of the medium from left to right. Positive  $Y_m$  values begin at the origin and increase along the left side of the medium from top to bottom. The media origin used by a printer can be affected by how forms are fed through the printer. To determine the media origin used by your printer, refer to *Printing Systems: Printer Information*.

In AFP, data is positioned on a logical page relative to the logical page origin. The logical page is positioned on the medium by aligning the top of the logical page with the top of the medium, and positioning the logical page origin relative to the media origin. That is, the logical page does not rotate; the objects and text within the logical page rotate relative to the logical page origin. Figure 5 on page 9 illustrates the relationship between the coordinate systems. This relative location, often called the *logical page offset*, is contained in a *form definition*. A form definition is an AFP resource that is used to specify formatting information about the physical form, such as overlays to be used, text suppression, and the position of page data on the form. It is specified when the document is submitted for printing or opened for viewing.



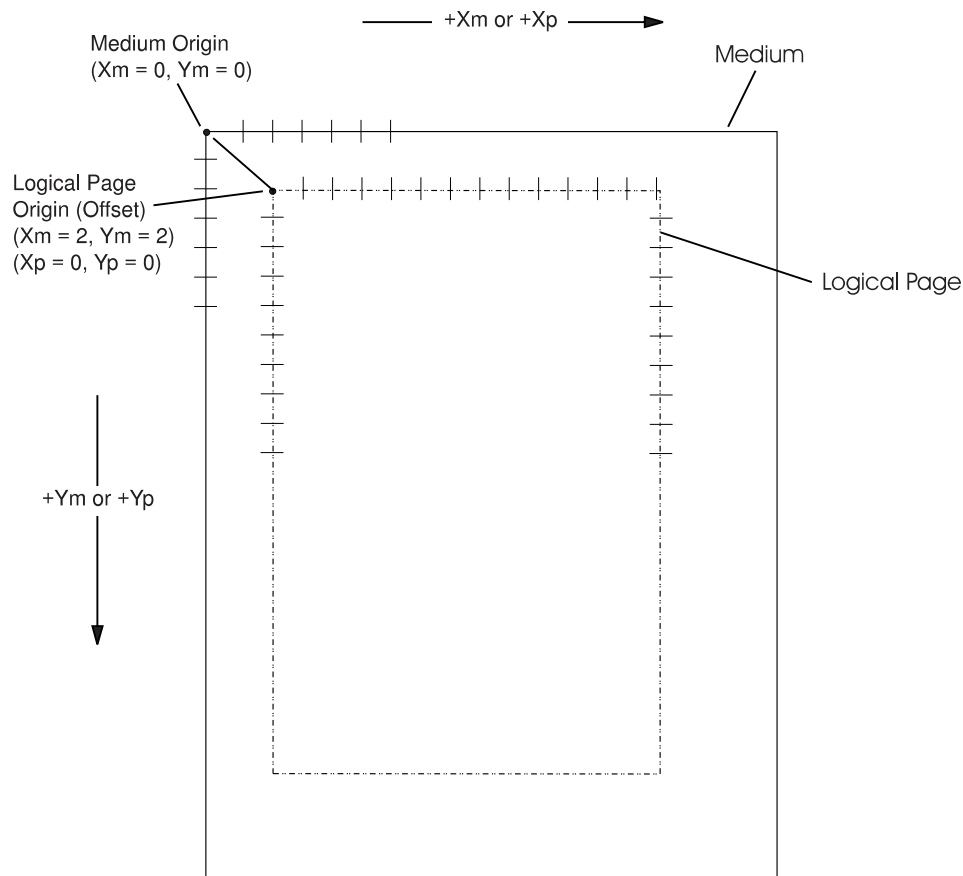


Figure 5. Relationship between the Logical Page and Media Coordinate Systems

Each logical page does not need to contain all of the data that will ultimately be presented on a medium. The page can refer to external files (AFP resources), which AFP presentation programs can retrieve when the document is submitted for printing or opened for viewing. The next section describes these AFP resources.

## AFP Data and Resource Objects

An AFP data object contains a single type of presentation data; that is, either presentation text, vector graphics, raster images, or bar codes, plus all the controls required to present the data. Applications can generate some types of data as AFP data objects, and then include these data objects in their output. Programs can format each data object separately from any other data object on a page, or programs can also present multiple objects of the same or different types in any sequence on a page.

An AFP resource object is a collection of presentation instructions and data consisting entirely of AFP structured fields. Resource objects are referenced by name in the presentation data stream and can be stored in system libraries in order for multiple applications and the print server to use them. Some types of resource objects can also be included (inline) in the presentation file, so that library access is not required when printing or viewing the file.

Resources are created with other applications and placed in a library. When you include a resource in your document, AFP Toolbox builds a reference to the resource that the print server later integrates with the AFP Toolbox output. You can use products such as these to create resources:

- **Overlays**
  - Overlay Generation Language/390 (OGL/390)
  - IBM AFPDS Windows® Driver

- FormsDesigner by ISIS Information Systems, Inc.
- ElixirForm by Elixir Technologies Corporation
- AFP Workbench for Windows can create overlays from viewed AFP pages.
- Infoprint® Server for OS/390, PCL2AFP and PS2AFP transforms (page segments)
- Infoprint Manager for AIX and Infoprint Manager for Windows NT® and Windows 2000, PCL2AFP and PS2AFP transforms (page segments)
- **Form definitions**
  - Page Printer Formatting Aid/390 (PPFA/390)
  - Application Builder for AFP by Elixir Technologies Corporation
  - FormsDesigner by ISIS Information Systems, Inc.
- **Page segments, images, and graphics**
  - Graphical Data Display Manager (GDDM)
  - IBM AFP Printer Drivers for Windows
  - Infoprint Designer for iSeries
  - Infoprint Server for OS/390, PCL2AFP and PS2AFP transforms (page segments)
  - Infoprint Server for iSeries, JPEG2AFP, GIF2AFP, and TIFF2AFP transforms (images)
  - Infoprint Manager on AIX and NT, PCL2AFP and PS2AFP transforms (page segments)

There are also many third party products that can create these resources.

## AFP Data Objects

The AFP data objects created by the AFP Toolbox are:

- Image objects (in Image Object Content Architecture (IOCA) that contain raster data like that produced by a facsimile or scanning device)
- Text objects
- Bar code objects (in Bar Code Object Content Architecture (BCOCA) format)
- Graphic objects (in Graphic Object Content Architecture (GOCA) format)

You can also include prebuilt resources and other objects in an AFP document. See Include Object, Include Overlay, and Include Page Segment in Chapter 3, “AFP Toolbox Language Reference” on page 53 for more information.

In addition to creating and referencing data objects, AFP Toolbox can read and include image and graphics objects inline in the output document.

## AFP Resource Objects

The five types of AFP resource objects are:

- Fonts
- Page segments
- Overlays
- Form definitions
- Page definitions

The following sections provide additional information about using these resources.

### Fonts

Fonts are collections of graphics characters of a given size and style used to present text.

Fonts present text characters requested for presentation in a page, an overlay, or a page segment. Local font identifiers are used in the data stream to select specific fonts. These identifiers are matched to the resource files containing the actual font data.

## Page Segments

Page segments are collections of image graphics or text data objects that can be presented at any location on a page. Examples of items that can be page segments include logos, signatures, bar charts, and engineering drawings.

Page segments can contain a mixture of text, image objects, and graphics data objects, and can be placed anywhere on a presentation page. Programs can request page segments for presentation in a page or overlay. Because page segments inherit the environment (such as the local font identifiers) defined by the page or overlay that includes them, you can think of them as pieces of pages. Page segments are used for logos, signatures, and boilerplate.

## Overlays

Overlays are collections of predefined data objects, such as boxes, lines, shading, text, logos, and graphics, that can be merged with application data for presentation. Overlays are often used as electronic forms.

Overlays can contain a mixture of text, image, graphics, and bar code data objects, and also can include page segments. Unlike page segments, overlays contain all of the environment information required for their presentation.

AFP defines two types of overlays: medium overlays and page overlays.

- Medium overlays are referenced by a form definition, and are positioned on the medium relative to the media origin.
- Page overlays are referenced by a page and are positioned on the medium relative to the page origin.

## Form Definitions

Form definitions contain information that defines the presentation of the page on the medium, such as where the page should be placed on the medium and whether the data should be printed on one or both sides of the paper. Refer to *IBM Page Printer Formatting Aid: User's Guide* for more information about form definitions.

A form definition can request that a medium overlay be presented on the medium with the page of data. You must use a form definition whenever you print or display an AFP document.

Each form definition contains one or more copy groups, which can be changed between pages of a document to dynamically select the form-mapping controls for subsequent pages. Using copy groups (also called medium maps), the application can accomplish the following:

- Position the logical page on the physical medium
- Print on both sides of a sheet of paper (duplex printing)
- Include medium overlays
- Select the number of copies of any page of data (to replace traditional multipart forms)
- Suppress selected fields (to replace the use of spot carbons)
- Specify offset stacking of cut-sheet output or marking the edges of continuous-forms output
- Select among input bins on a printer
- Select the level of print quality
- Specify the page presentation (portrait or landscape)
- Specify that multiple pages of data are to print on a single side of a form by defining partitions of the form.

## Page Definitions

Page definitions contain instructions for formatting data intended for a line printer into pages that can be printed on an AFP printer. Refer to *IBM Page Printer Formatting Aid: User's Guide* for more information about page definitions.

A page definition is used whenever you print a line-data file on an AFP printer, however it is not required when you print or display an AFP file. With AFP Toolbox, page definitions are not necessary since the AFP Toolbox produces an AFP file rather than a line-data file. (The information contained in the page definition is already contained in the output that AFP Toolbox produces from the application.)

---

## Grouping, Archiving, and Linking Functions

The AFP Toolbox provides you with the capability to group together different types of information, and to add hypertext links to information for use with AFP Workbench and OnDemand.

**Note:** Refer to *IBM OnDemand for AIX: Introduction and Planning Guide* for more information about using OnDemand and an archival and retrieval tool for your AFP information.

Documents designed for viewing on a workstation should contain indexing information to facilitate navigating through the document. Archival and retrieval applications can use this indexing information to identify separate parts of a large document for saving or restoring. This section describes the grouping and linking functions that are supported.

### The Grouping Function

In order to illustrate the grouping function, use a bank statement application as an example. The application consists of thousands of individual customer statements. You can think of each of these statements as a smaller separate unit, each of which is uniquely identified by an account number or other attributes such as date and Social Security number. AFP Toolbox lets you divide the large bank statement application into individual units, called groups, by inserting structured fields in the file that define the group boundaries. In other words, a group is a named collection of sequential pages, which in this application, consists of the pages describing a single customer's account.

AFP Toolbox provides procedure calls that generate indexing information, with which you can accomplish these tasks:

- Define boundaries within documents (called *page groups*) in the AFP Toolbox output document. For example, the first and last pages of the statement for each customer in the file containing many customer statements might define the boundaries of a page group.
- Identify a group of pages with an indexing tag containing an attribute name and value, for example, an Account Number attribute associated with the account number of each customer. This type of tag is called a *group-level* tag since it is associated with a group of pages.
- Identify a single page with an indexing tag containing an attribute name and value pair, for example, a Total Charges attribute associated with the charges for each specific customer. This type of tag is called a *page-level* tag since it is associated with a single page.

Using AFP Workbench, you can locate a group of pages using the indexing attribute names and values defined for each group. You can navigate through a large file to locate a single customer statement more quickly than by performing a string search on, for example, a customer's name.

### The Linking Function

You might want to go from one part of an online document to another or from one document to another document while viewing it. Searching through the document or finding and opening the other document is time consuming. By using AFP Toolbox functions, you can add hypertext links that insert structured fields to define a link launch point (where you want the link to start), and target point (where you want the link to go).

---

## Chapter 2. Using AFP Toolbox

This chapter describes how to use AFP Toolbox with C and C++ to generate the document shown in Figure 6 on page 14. Samples for COBOL and RPG are also provided in the appendixes for your operating system. Source for C, C++, and COBOL samples are shipped with AFP Toolbox on the appropriate operating system. You can use them as templates for your own applications.

This chapter also gives examples using bar codes and tables with AFP Toolbox and describes how to use fonts with this product.

The sample program creates a bank statement. The program reads a data file that contains transaction records during the statement period for each account that a customer has.

The output file is indexed based on the account numbers, customer name, address, and dates.

The sample reads the data file, and for each new customer does the following:

1. Builds a customer record containing the name, address, account numbers, and a list of savings and checking account transactions.
2. Creates a new bank statement page and includes the bank logo, name, and address.
3. Puts indexing information in the output data stream for the customer.
4. Writes the box and headings for the columns that contain the account information (Date, Transaction, and so forth).
5. Places the "this is a sample" footer at the bottom of the page.
6. Processes the savings account transactions (sorted by date).
7. As each transaction is processed, it checks to see if the current position has reached the bottom of the page. When it is within one inch of the bottom, it creates a new page and counts it.
8. When all of the transactions are finished, it returns to the first page and adds "Page x of y" to each page.
9. Places the pages into the document.

## Sample Document

**Wombat Savings and Loan**

Wombat Savings and Loan  
100 1st. Street  
Prestige, FL 03636

JENNY A. BOWMAN  
2243 WILDERNESS LN. #4B  
GREENVILLE, FL 03606

Savings Account: 32533500.1  
Checking Account: 32533500.2

Statement From: Dec 20 1994  
to: Jan 19 1995

Page 1 of 2

Date	Transaction	Deposit	Withdrawal	Balance
<i>Savings Account: 32533500.1</i>				
Dec 20 1994	Opening Balance			\$ 770.21
Dec 21 1994	Deposit	33.85		804.06
Jan 11 1995	Deposit	12.77		816.83
Jan 19 1995	Dividend	8.31		825.14
Jan 19 1995	Closing Balance			\$ 825.14
<i>Checking Account: 32533500.2</i>				
Dec 20 1994	Opening Balance			\$ 1266.96
Dec 20 1994	Draft 1350		12.20	1254.76
Dec 20 1994	Draft 1345		13.54	1241.22
Dec 21 1994	Draft 1355		9.99	1231.23
Dec 22 1994	Deposit	288.14		1519.37
Dec 22 1994	Draft 1359		7.60	1511.77
Dec 22 1994	Draft 1348		81.00	1430.77
Dec 22 1994	Draft 1356		110.93	1319.84
Dec 23 1994	Draft 1349		17.75	1302.09
Dec 23 1994	Draft 1353		76.34	1225.75
Jan 1 1995	Draft 1347		5.67	1220.08
Jan 1 1995	Draft 1358		283.56	936.52
Jan 1 1995	Debit Card Annual Fee		12.00	924.52
Jan 2 1995	Draft 1360		12.76	911.76
Jan 4 1995	Deposit	197.29		1109.05
Jan 4 1995	Draft 1352		74.65	1034.40
Jan 10 1995	Deposit	45.39		1079.79
Jan 10 1995	Draft 1357		8.95	1070.84
Jan 10 1995	Draft 1354		26.57	1044.27
Jan 11 1995	Draft 1346		13.89	1030.38
Jan 12 1995	Draft 1361		22.95	1007.43
Jan 12 1995	Deposit	431.09		1438.52
Jan 12 1995	Draft 1363		110.57	1327.95
Jan 13 1995	Deposit	9.35		1337.30

AFP Toolbox Sample Bank Statement Application Output

- 1 Strings of text, aligned in various ways.
- 2 Included page segment.
- 3 Boxes containing text strings.
- 4 Drawing a rule.
- 5 Columns of text.
- 6 Determining page breaks.
- 7 Counting page "x" of "y".

Figure 6. Sample Document

---

## C Sample Notes

The sample C application describes the use of handles. Handles are required for the C functions to identify the session and piece of the documents being built by AFP Toolbox. Refer to “Understanding Handles” on page 59 for additional information.

Handles are always returned by a “Begin” function for an object, and used by subsequent calls to place data into the object. The handle tells the API where the particular data belongs.

For example, when you issue the Begin Document function call, the function returns a document handle. You use this handle on subsequent Begin Page calls to indicate which document the page belongs in. Similarly, Begin Page returns a page handle used on calls such as Write String to specify which page the text string goes in. In this manner, you can have many documents or pages being built at one time. There is no limit on the number of documents or pages that can be “open” at once, although each page will consume a fair amount of storage as it is being built.

---

## C++ Sample Notes

The C++ sample program is not written in true object-oriented fashion and is not intended as a sample of object-oriented programming. Rather, it is a transformation of the sample C program to use the C++ interface of AFP Toolbox to demonstrate the differences between the two interfaces. It also provides a common frame of reference for learning AFP Toolbox interfaces.

By using object-oriented programming techniques, identifying where inheritance could be used, and taking advantage of the similarity of input record processing, much could be done to reduce the size and complexity of this program.

---

## COBOL Sample Notes

The COBOL samples in this book assume you are using the PERFORM statements and variables provided with the toolbox in the ATXCBPRF and ATXCBVAR copy books. If you choose not to use the IBM-supplied PERFORM statements then you will need to declare your own variables. Make sure that they have the same PICTURE clause as the variables declared in ATXCBVAR.

---

## Getting Started

The approach to building a document is:

1. Start your session with the Begin Session function call (not C++). The session initialization must successfully complete before you invoke any other call. See “Beginning the Session” on page 16.
2. Define the fonts used in the application. See “Defining Fonts” on page 17.
3. Begin a document. See “Beginning a Document” on page 18.
4. Specify indexing information. See “Indexing” on page 19.
5. Create a page objects (C++ only). See “Creating Page Objects” on page 20.
6. Begin a page. See “Beginning a Page” on page 21.
7. Put data on the page. In the example, data consists of strings of text, horizontal rules, and a page segment (artwork). See “Putting Data on the Page” on page 22.
8. Check to see if the page is filled and start a new page as necessary (optional). See “Determining Page Breaks” on page 28.
9. End the page. See “Ending a Page” on page 29.
10. Repeat 6 through 9 until the document is complete.
11. End the document. See “Ending a Document” on page 31.
12. End AFP Toolbox session (C interface only). See “Ending the Session” on page 32.

## Getting Started

### Notes:

1. You can build more than one page at a time as the sample shows.
2. There is more information about each of these calls in Chapter 3, “AFP Toolbox Language Reference” on page 53.

## Beginning the Session

This call establishes AFP Toolbox session and must successfully complete before you invoke any other call. This is not done when using C++. AFP Toolbox returns a handle which must be used in some subsequent calls to AFP Toolbox in this session, such as AFPBgnDoc. See “C Sample Notes” on page 15 for a description of handles.

- **C Application**

```
int rc;                                /* return code */
TBHANDLE toolboxSession;               /* session handle */
rc = AFPBgnSession(&toolboxSession);

/*****
/* call function to check for non-zero return code. */
/* if other than zero rc, checkForError will end the */
/* program and display the message you pass it.      */
*****/

checkForError(rc, "Error Opening AFP Toolbox Session");
```

- **COBOL interface:**

```
PERFORM AFPINIT.
```



## Defining Fonts

Define and specify the fonts to use to print character strings. You can define all the fonts in your document here and then refer to them as needed with SetFont calls. The font is selected by the Set Font method. These samples show the font for the bank name and address being defined and set. They specify a 14 point, bold, Times New Roman Latin1 typeface with code page 500 (T1V10500) for the output.

- **C Application**

```
FontID titleFont;
char outputCodepage[] = "T1V10500";
rc = AFPDefineFontByAttr(outputCodepage, /* output code page ID */
                        "TIMES NEW ROMAN LATIN1", /* typeface */
                        140, /* point size times 10 */
                        BOLD_WT, /* bold font */
                        MEDIUM_WD, /* normal width */
                        NORMAL_FS, /* not italic */
                        &titleFont); /* address for font ID */
checkForError(rc,"Error Defining Address Font");

...

AFPSetFont(page, /* page handle returned from AFPBgnPage */
           titleFont); /* font ID returned from AFPDefineFontAttr */
```

- **C++ Application**

```
FontID titleFont;
char outputCodepage[] = "T1V10500";
rc = AFPDefineFontAttr(outputCodepage, /* output code page ID */
                      "TIMES NEW ROMAN LATIN1", /* typeface */
                      140, /* point size times 10 */
                      BOLD_WT, /* bold font */
                      MEDIUM_WD, /* normal width */
                      NORMAL_FS, /* not italic */
                      &titleFont); /* address for font ID */

...

textObject->SetFont(titleFont); /* This is a Ptext object method.
                                /* Ptext was created by the
                                /* constructor for the FNBPage.
```

- **COBOL Interface:**

```
MOVE "T1V10500" TO AFP-CODE-PAGE.
MOVE "TIMES NEW ROMAN LATIN1" TO AFP-DESCRIPTIVE-NAME.
MOVE 140 TO AFP-POINT-SIZE.
MOVE BOLD TO AFP-WEIGHT.
MOVE NORML TO AFP-FONT-WIDTH.
MOVE ROMAN TO AFP-STYLE.
DISPLAY "FONT 1".
PERFORM AFPDFNTAT.
MOVE AFP-FONT-ID TO TIM14MED.
```

## Begin Document

### Beginning a Document

You must begin and end every document.

This call begins a document and specifies the output document name and type, where the output is placed, and the name of the output file.

- **C Application**

```
TBHANDLE outputDocument;          /* document handle */
char outputDocName[] = "BANKSAMP";
char outputDocComment[] = "AFP Toolbox Bank Statement "
                          "Sample Application Output Document";
char outputDocFileName[] = "ctest1.afp";

rc = AFPBgnDoc(toolboxSession,    /* session handle from AFPBgnSession */
               outputDocName,     /* name of document in the output */
               outputDocComment,  /* comment to be placed in output */
               TO_OUTPUT_FILE,    /* output goes to a file */
               outputDocFileName, /* name of the output file */
               NULL,              /* place holder for output buffer */
               &outputDocument); /* address of HANDLE to be returned */

checkForError(rc, "Error Opening Output Document");
```

- **C++ Application**

```
char outputDocName[] = "BANKSAMP";
char outputDocComment[] = "AFP Toolbox Bank Statement "
                          "Sample Application Output Document";
char outputDocFileName[] = "cpptest1.afp";
#ifdef UNIX
#define OPEN_FLAGS ios::out
#else
#define OPEN_FLAGS ios::out | ios::binary
#endif

ofstream os(outputDocFileName, OPEN_FLAGS);
AFPDoc* outputDocument;
outputDocument = new AFPDoc(outputDocName,    // Name for output
                           outputDocComment, // Comment for output
                           os);              // Name of fstream object
                                              // (this stores the output file)
```

- **COBOL Interface:**

```
MOVE "SUPER SUN SEEDS" TO AFP-DOC-NAME.
MOVE "MVS COBOL Program" TO AFP-DOC-COMMENT.
MOVE FILED TO AFP-OUTPUT-TYPE.
MOVE "FILEOUT" TO AFP-OUTPUT-FILENAME.
PERFORM AFPBDOC.
```

## Indexing

The sample program includes data stream indexing so that information in it can be processed by applications such as AFP Workbench. For more information about indexing, see “Grouping, Archiving, and Linking Functions” on page 12.

In the sample, the information for each customer in the output is indexed on customer name, account number, address, and opening and closing dates of the statement.

This portion of sample code is executed at the beginning of each new customer statement. The pages belonging to the statement are marked with Begin Group to indicate they go together using the customer name as the group name. The program then extracts the necessary information from the customer record and builds the indexes using Put Tag. These indexes apply to the entire group of pages. After the statement is finished and all pages have been placed in the output, use End Group to mark the end of the statement.

- **C Application**

```
rc = AFPBgnGroup(outputDocument, cr->name);
rc = AFPPutTag(outputDocument, "Customer Name",
               cr->name);
rc = AFPPutTag(outputDocument, "Customer Address",
               cr->address);
rc = AFPPutTag(outputDocument, "Customer City State Zip",
               cr->cityStatezip);
rc = AFPPutTag(outputDocument, "Savings Account Number",
               cr->savingsAccount);
rc = AFPPutTag(outputDocument, "Checking Account Number",
               cr->checkingAccount);
rc = AFPPutTag(outputDocument, "Opening Date",
               cr->openingDate);
rc = AFPPutTag(outputDocument, "Closing Date",
               cr->closingDate);

...
rc = AFPEndGroup(outputDocument);
```

- **C++ Application**

```
outputDocument.BeginGroup(cr->name); //Identifies these pages as belonging to
                                   //a group. The group name is the customer name.
outputDocument.Tag("Customer Name",cr->name); //The Tag method tells
outputDocument.Tag("Customer Address",cr->address); //Toolbox how to build
outputDocument.Tag("Customer City State Zip",cr->cityStateZip); //the index. Toolbox
outputDocument.Tag("Savings Account Number",cr->savingsAccount); //gathers information,
outputDocument.Tag("Checking Account Number",cr->checkingAccount); //such as the name then
outputDocument.Tag("Opening Date",cr->openingDate); //labels it with a string
outputDocument.Tag("Closing Date",cr->closingDate); //such as Customer Name.
...
outputDocument.EndGroup(); //Identifies the end of the group.
```

- **COBOL Interface**

```
MOVE "MY GROUP" TO AFP-GROUP-NAME.
PERFORM AFPBGRP.
SET AFP-CURRENT-HANDLE TO AFP-DOCUMENT-HANDLE.
MOVE "GROUP TAG VALUE" TO AFP-TAG-VALUE.
MOVE "MY TAG NAME" TO AFP-TAG-NAME.
PERFORM AFPPTAG.
...
PERFORM AFPEGGRP.
```

## Create Page Objects

### Creating Page Objects

When using C++, you create page objects for each page in the document. The sample defines a class called FNBPage which is derived from AFPPage. The constructor for FNBPage takes as input a customer record (CustomerRecord\*) which was built by the application as it read in the customer data.

Pages consist of text and image objects. The text, rules, and boxes for each page are built in a presentation text object (Ptext) (refer to “PText” on page 326 for background information) by the FNBPage constructor. The first few lines of the constructor are shown so you can see how this is done.

In this example, the data in the customer record is processed and the constant pieces of the page are created. These pieces are: the Wombat Savings and Loan logo, the bank name and address, the customer’s name and address, and the header boxes for the columns of transaction data. You could use a table to make the header boxes. See “Working with Tables” on page 36 for more information.

- **C++ Application**

```
class FNBPage : public AFPPage
{
public:
    FNBPage(CustomerRecord* cr);
    ~FNBPage() {
        if (textObject) delete textObject;
    }
    PText* text(void) {
        return textObject;
    }
    void finished(void) {
        IncludeText(*textObject);
    }

private:
    PText* textObject;
};

FNBPage::FNBPage(CustomerRecord* cr) :
AFPPage(cr->name),    // page name is from the customer record
textObject(0)        // no text object yet
{
    textObject = new PText(cr->name); // create text object for the page
    ...
}
```

## Beginning a Page

You must begin and end every page within the document. AFP Toolbox will not automatically end pages when they become full; page ejects need to be done by your application. You can build more than one page at a time, but the MO:DCA data stream for a page is not produced until the page is ended.

This call begins a page within a document and provides the name to be used in the output.

In the C++ sample, FNBPages are built for each customer as shown on “Creating Page Objects” on page 20. In your own applications you can define a special class for pages, like the sample does, or you can use the AFPPage class supplied with AFP Toolbox.

This excerpt shows the first page of each statement being built.

- **C Application**

```
TBHANDLE page;
rc = AFPBgnPage(outputDocument, /* handle from AFPBgnDoc */
                cr->name,        /* page name from customer data */
                &page);          /* address of HANDLE to return */
```

- **C++ Application**

```
FNBPage* currentpage, *firstpage;
firstpage = new FNBPage(cr); // input is the customer record
```

- **COBOL Interface**

```
PERFORM AFPBPAGE.
SET PAGE1-HANDLE TO AFP-PAGE-HANDLE.
```

## Putting Data on the Page

### Putting Data on the Page

After you begin the document and begin a page, you can use various functions to put data into the page. Before putting data on a page, you must specify its position on the page with the various calls provided and then start the appropriate font for the text. You can also specify an input and output code page. For information about code pages, see “Working with Code Pages” on page 44. You can put these types of data on a page:

Strings of text, see page 22.

Rules, see page 24.

Boxes, see page 24.

Resources (art and overlays), see page 25.

Paragraphs (not available in C<sup>++</sup>), see page 26.

Data objects (not shown in the example). See page 33 for information about including bar codes.

### Strings of Text

A text string is printed exactly as entered (that is, with no formatting except alignment). The example has several text strings that are aligned differently. For example, the bank name and address are left aligned, the phrases “Statement from:” and “to:” are right aligned, and the columns of numbers are character aligned on the decimal point.

The text strings are identified with **1** and **5** in Figure 6 on page 14. Text strings can be placed in a page with the following alignments relative to the current position:

**Left** The first character is at the current position.

```
Current position = ↓
                  A B C
                  A B C
```

**Right** The last character is at the current position.

```
Current position = ↓
                  A B C
                  A B C
```

**Center**

Characters are centered around the current position.

```
Current position = ↓
                  A B C
                  A B C
```

**Character**

A specified character is placed at the current position.

```
Current position = ↓
                  1 2 3 .
                  .1 2 3
```

**Note:** If you want a string to use a different font from the one just used, you must define the font before creating the string. See “Defining Fonts” on page 17 for instructions.

The following segment shows how to create the opening and closing dates of the statement. Other strings are placed in a similar manner, varying the font, position, and alignment parameters as appropriate.

**Creating a String:** The functions listed in Table 1 can be used to position the baseline of the character string:

Table 1. Positioning Functions

Function	Position
Horizontal Move	Sets the horizontal position relative to the current position.
Horizontal Move To	Sets the horizontal position to an absolute location.
Vertical Move	Sets the vertical position relative to the current position.

Table 1. Positioning Functions (continued)

Vertical Move To	Sets the vertical position to an absolute location.
Next Line	Moves down one line space in the current font.
Set Position	Sets both the horizontal and vertical positions to either a relative or absolute new location.

This example builds the lines that indicate the opening and closing dates of the statement. The phrases “Statement from:” and “to:” are built as a combination of right- and left-aligned text. Vertical Move and Horizontal Move are used to position the data. The COBOL example uses Set Position to place the data. The first line starts down 2.5 inches and over 6.75 inches on the page. Subsequent lines are positioned by Next Line, which advances one line in the current font. Next Line takes as input the desired horizontal position; in this case, five inches. There is no font change between the first and second line of text.

#### • C Application

```

AFPVMoveTo(page, IN_2_1440(2.5)); /* set vertical position */
AFPHMoveTo(page, IN_2_1440(6.75)); /* set horizontal position */
char buffer[15];
AFPWriteString(page, "Statement From: ", /* text string */
    DEFAULT_WS, /* use default line spacing */
    DEFAULT_ICS, /* use default character spacing */
    RIGHT_ALIGN, /* text is right aligned */
    ' ', /* character for alignment (ignored) */
    FALSE); /* do not leave the position set */
datestr = ctime(&cr->openingDate); /* get date from cust record*/
memcpy(buffer, &datestr[4], 7); /* copy to buffer */
memcpy(&buffer[7], &datestr[20], 20);
buffer[11] = '/0'; /* null terminate the string */

AFPWriteString(page, buffer, /* date is now in the buffer */
    DEFAULT_WS, /* use default line spacing */
    DEFAULT_ICS, /* use default character spacing */
    LEFT_ALIGN, /* text is left aligned */
    ' ', /* character for alignment (ignored) */
    FALSE); /* do not leave the position set */
AFPNextLine(page); /* move down one line space */

AFPWriteString(page, "to: ", -1, -1, RIGHT_ALIGN, FALSE); /*FALSE indicates that the
/*horizontal position is not
/*moved to the right end of
/*the string after it is printed.

datestr = ctime(&cr->closingDate);
memcpy(buffer, &datestr[4], 7);
memcpy(&buffer[7], &datestr[20], 20);
buffer[11] = '/0';
AFPWriteString(page, buffer, -1, -1, LEFT_ALIGN, FALSE);

```

#### • C++ Application

```

textObject->MoveTo(IN_2_U1440(6.75), // horizontal pos 6.75 inches
    IN_2_U1440(2.5); // vertical pos 2.5 inches
textObject->AddText("Statement from: ", // text string
    -1, // use default line spacing
    -1, // use default character spacing
    RIGHT_ALIGN, // text is right aligned
    ' ', 0); // character for alignment
datestr = ctime(&cr->openingDate);
memcpy(buffer, &datestr[4], 7);
memcpy(&buffer[7], &datestr[20], 20);
buffer[11] = '/0';
textObject->AddText(buffer); // place the date string
textObject->NextLine(IN_2_U1440(6.75)); // advance one line

textObject->AddText("to: ", -1, -1, LEFT_ALIGN, ' ', 0);
datestr = ctime(&cr->closingDate);

```

## Putting Data on the Page

```
memcpy(buffer, &datestr[4], 7);
memcpy(&buffer[7], &datestr[20], 20);
buffer[11] = '/0';
textObject->AddText(buffer);           // Place the date string on the page.
```

- **COBOL interface**

```
MOVE MM TO AFP-UNIT-OF-MEASURE
PERFORM AFPSUNI.
MOVE 10 TO AFP-X-COORDINATE.
MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
MOVE 20 TO AFP-Y-COORDINATE.
MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
PERFORM AFPSPOS.
MOVE 20 TO AFP-STRING-LENGTH.
MOVE "line of text on page" TO AFP-CHARACTER-STRING.
MOVE LEFT-ALIGN TO AFP-ALIGNMENT-OPTION.
MOVE FALS TO AFP-POSITION-OPTION.
PERFORM AFPWRITE.
PERFORM AFPNLINE.
```

## Rules

You can draw vertical and horizontal rules with AFP Toolbox. C++ methods support fixed length rules, while C functions let you draw fixed and variable size rules.

The rule is identified with **4** in Figure 6 on page 14.

The following samples show how to draw the rule in the statement header:

Draw a horizontal rule to the right of the current location, 7.5 inches long and 6/1440ths of an inch (.1 inches for COBOL) thick. In C, you can draw variable size rules with Begin Rule and End Rule.

- **C Application**

```
AFPPutHRule(currentpage, IN_2_U1440(7.5), 6);
```

- **C++ Application**

```
text->DrawRule(6, IN_2_U1440(7.5));
```

- **COBOL Interface**

```
MOVE IN TO AFP-UNIT-OF-MEASURE
PERFORM AFPSUNI.
MOVE 7.5 TO AFP-RULE-LENGTH.
MOVE .1 TO AFP-RULE-THICKNESS.
PERFORM AFPPHRUL.
```

## Boxes

You can draw boxes with AFP Toolbox. C++ methods support fixed length boxes, while C, COBOL, and RPG functions let you draw fixed and variable size boxes.

The boxes are identified with **3** in Figure 6 on page 14.

This sample draws the boxes that contain the headings of the statement. This is drawn as one large box with vertical rules dividing it into five smaller boxes. The sample shows a fixed size box. In C, you can use the Begin Box and End Box functions to draw a variable size box. You could use a table instead of boxes to create the headings in a box. For more information, see “Working with Tables” on page 36.

- **C Application**

```
AFPHMoveTo(page, IN_2_U1440(.5)); /* Set position for box */
AFPVMoveTo(page, IN_2_U1440(4));
AFPPutBox(page, 6, IN_2_U1440(7.5), /* Box is 7.5 inches long */
          IN_2_U1440(.25)); /* and .25 inches deep */
AFPHMove(page, IN_2_U1440(1)); /* Move over 1 inch */
AFPPutVRule(page, IN_2_U1440(.25), 6); /* Draw vertical rule */
AFPHMove(page, IN_2_U1440(3.5)); /* Move over 3.5 inches */
AFPPutVRule(page, IN_2_U1440(.25), 6); /* Draw vertical rule */
```



```

AFPHMove(page,IN_2_U1440(1));      /* Move over 1 inch      */
AFPVRule(page,IN_2_U1440(.25),6); /* Draw vertical rule    */
AFPVRule(page,IN_2_U1440(1));      /* Mmove over 1 inch     */
AFPVRule(page,IN_2_U1440(.25),6); /* Draw vertical rule    */

```

- **C++ Application**

```

textObject->DrawBox(6,              // rule width for box sides
                    IN_2_U1440(.5),  // starting horizontal pos
                    IN_2_U1440(4),    // starting vertical pos
                    IN_2_U1440(7.5), // width of box
                    IN_2_U1440(.25)); // depth of box
textObject->HMove(IN_2_U1440(1));    // move over 1 inch
textObject->DrawRule(IN_2_U1440(.25),6); // draw vertical rule
textObject->HMove(IN_2_U1440(3.5));  // move over 3.5 inches
textObject->DrawRule(IN_2_U1440(.25),6); // draw vertical rule
textObject->HMove(IN_2_U1440(1));    // move over 1 inch
textObject->DrawRule(IN_2_U1440(.25),6); // draw vertical rule
textObject->HMove(IN_2_U1440(1));    // move over 1 inch
textObject->DrawRule(IN_2_U1440(.25),6); // draw vertical rule

```

Draw a box 7.5 inches wide and .25 inches deep with lines with rules 6 1440ths of an inch thick.

- **COBOL Interface**

```

MOVE 6 TO AFP-RULE-THICKNESS
MOVE IN TO AFP-UNIT-OF-MEASURE
PERFORM AFPSUNI.
MOVE 7.5 TO AFP-BOX-WIDTH
MOVE .25 TO AFP-BOX-DEPTH
PERFORM AFPPBOX.

```

## Resources

Resources are objects that are created with other applications and placed in a library. AFP Toolbox builds a reference to the resource that the print server later integrates with AFP Toolbox output. Page segments and overlays are examples of resources. All referenced resources must be available to AFP Toolbox, the print server, or both.

**Including a Resource:** Follow these steps to place a resource (object, overlay, or page segment) on a page:

1. Set the position for the upper-left corner of the resource.
2. Put the resource at the specified position.

You will get an error at print time, but not during AFP Toolbox execution, for either of the following conditions:

- If you reference a resource that does not exist or is in a resource library that is not available to PSF.
- If you place an overlay, page segment, or object at a valid position on the page, but the object is too big to fit on the page.

The following sample code shows how to position and include a page segment. The page segment is identified with **2** in Figure 6 on page 14.

Set the position for the page segment to be down .5 inches and over from the left .5 inches on the logical page. Include the page segment named WSLLOGO and indicate that it is to be used multiple times in the document.

- **C Application**

```

AFPVMoveTo(page,IN_2_U1440(.5));
AFPVMoveTo(page,IN_2_U1440(.5));

AFPIncIPSeg(page, "WSLLOGO",TRUE); /*TRUE specifies that the page segment is
/*used more than once.

```

- **C++ Application**

## Putting Data on the Page

```
currentpage->IncludePageSegment("WSLLOGO",    // page segment name
                                IN_2_U1440(.5), // horizontal position
                                IN_2_U1440(.5), // vertical position
                                HIGH_UF);       // frequency; HIGH_UF indicates
                                                // that the page segment
                                                // is used more than once.
```

Include the page segment INVHEAD1 and indicate that it is to be used multiple times in the document.

- **COBOL Interface**

```
MOVE "INVHEAD1" TO AFP-PSEG-NAME.
MOVE TRU TO AFP-REUSE-OPTION.
PERFORM AFPIPSEG.
```

### ***Other AFP Toolbox Functions that Use Resources:***

#### **Invoke Medium Map:**

References a medium map (also called a copy group) from the form definition being used for printing AFP Toolbox output. You can use this function between pages to switch copy groups.

#### **Include Object:**

References an image, graphic, or other type of object for inclusion at print time.

#### **Include Page Overlay:**

References an overlay for inclusion at print time.

#### **Include Page Segment:**

References the page segment for the printer to include in the document at print time.

For information about using these functions, see Chapter 5, "AFP Toolbox C++ Object Library" on page 309 and Chapter 3, "AFP Toolbox Language Reference" on page 53.

## Paragraphs

Paragraphs are not used in the sample program but are available for use by C, COBOL, and RPG programs. Paragraphs are blocks (rectangles) of text. You specify how wide and deep you want the block to be and AFP Toolbox formats the text into those dimensions. You control the layout and appearance of the text in the paragraph by specifying the text alignment, the fonts used, and the line spacing.

Follow these steps to place a paragraph on a page:

1. Set the position for the top-left corner of the paragraph.
2. Begin the paragraph using Begin Paragraph. This function lets you specify the width of the paragraph, the alignment of text (left, center, right, or justified), the spacing between lines of output, and the maximum depth of the paragraph.
3. Put text in the paragraph using the Put Text function call. Each Put Text call specifies the font to be used for the following text.

**Note:** If the Put Text causes the maximum paragraph depth to be exceeded, you will receive return code 12 and the string that did not fit is returned in &remainingstr.

4. End the paragraph.

The most significant feature of a paragraph is that text flows within it. That is, words and sentences flow according to the formatting parameters set in the Begin Paragraph call regardless of how the words and sentences were entered on the Put Text calls.

- **C Application**

```
rc = AFPDefineFontAttr(currentdoc,cp500,"TIMES NEW ROMAN LATIN1", /* These calls
    pt12,MEDIUM_WT,MEDIUM_WD,NORMAL_FS,&f2);                    /* define and set
rc = AFPDefineFontAttr(currentdoc,cp500,"COURIER LATIN1",          /* two fonts that
    pt12,BOLD_WT,MEDIUM_WD,ITALIC_FS,&f4);                        /* can be used
AFPSetFont(currentpage,f4);                                         /* in the paragraph.
AFPSetPos(currentpage,240,ABSOLUTE,4*240,ABSOLUTE); /* Sets the position
AFPBgnPgraph(currentpage,IN_2_U1440(3), /* Begins a paragraph 3 inches wide
```

```

JUSTIFY_ALIGN, /* Justified text
264,          /* Fixed line spacing
IN_2_U1440(2)); /* Max depth 2 inches
rc = AFPPutText(currentpage,"The first sentence of this " /* The PutText calls
                        "justify-aligned paragraph " /* place the data
                        "uses font f2.",f2,&remainingstr);/* in the paragraph.
rc = AFPPutText(currentpage,"The second sentence of the " /* Check the return codes
                        "paragraph uses font f4.", /* to make sure the data
                        f4,&remainingstr); /* fits in the paragraph.
rc = AFPPutText(currentpage,"The third sentence of the "
                        "paragraph uses font f2.",
                        f2,&remainingstr);
rc = AFPPutText(currentpage,"The fourth sentence of the "
                        "paragraph uses font f4.",
                        f4,&remainingstr);
AFPPEndPgraph(currentpage); /*end the paragraph

```

The example uses justified alignment. You can also use these alignments: Left Aligned, Right Aligned, and Center Aligned.

- **COBOL Interface**

```

MOVE IN TO AFP-UNIT-OF-MEASURE.
PERFORM AFPSUNI.
MOVE 3.5 TO AFP-PARA-WIDTH.
MOVE LEFT-ALIGN TO AFP-FORMAT-OPTION.
MOVE DEFAULT-LSP TO AFP-LINE-SPACING.
MOVE 7.0 TO AFP-PARA-MAXDEPTH.
PERFORM AFPBPARA.
MOVE TIM10ITAL TO AFP-FONT-ID .
MOVE LOW-VALUES TO AFP-CHARACTER-STRING.
STRING AFP-STRING-1
    DELIMITED BY SIZE INTO AFP-CHARACTER-STRING.
CALL "STRING-LENGTH" USING AFP-CHARACTER-STRING,
    BY CONTENT LENGTH OF AFP-CHARACTER-STRING,
    BY REFERENCE AFP-STRING-LENGTH.
ADD 1 TO AFP-STRING-LENGTH.
PERFORM AFPPTTEXT.
PERFORM AFPEPARA.

```

## Page Breaks

### Determining Page Breaks

The sample programs show how to build multiple pages at the same time, determine when a page is full, count the number of pages, and print the string “Page x of y” on every page.

This portion of code from the sample application processes the transactions for a particular customer. After each transaction is placed, this code is executed.

It uses Query to determine the current y (vertical) position. Since Query returns other information, “dummy” parameters are passed as well.

Next the code checks to see if the y position is greater than 10 inches on the page. If so, a new page is started. The C example uses the create\_new\_page routine (see the sample code). Note that the current page is not ended because we still need to place the “Page x of y” string once we have counted all the pages.

- **C Application**

```
long yPos,dummy;
FontID dummy1;
MODCColors dummy2;

AFPQuery(page, &dummy, &yPos, &dummy2, &dummy1, &dummy); /*Get position

if (yPos > IN_2_U1440(10))
{
    page_handles[*numpages] =
        create_new_page(outputDocument, cr); /*Create new page
    (*numpages)++;                          /*Increment the page counter
    AFPSetFont(*currentpage,transactionFont);
}
```

- **C++ Application**

```
FNBPage* page = *currentpage;
PText* text = page->text();
short xPos, yPos;
text->QueryPosition(xPos, yPos); //Get position
if (yPos > IN_2_U1440(10))
{
    page = new FNBPage(cr); //Create a new page
    page_handles[*numpages] = *currentpage = page;
    text = page->text(); //Increment the page counter
    (*numpages)++;
    text->SetFont(transactionFont);
}
```

- **COBOL Interface**

```
MOVE IN TO AFP-UNIT-OF-MEASURE
PERFORM AFPSUNI.
PERFORM AFPNLINE.
PERFORM AFPQUERY.
IF AFP-RET-CODE = 13 OR AFP-Y-COORDINATE > 10.5
    PERFORM END-CUST-PAGE.
```

## Ending a Page

You must begin and end every page within the document. AFP Toolbox will not automatically end pages when they become full; page ejects need to be done by your application.

This call ends the page and places its contents into the document. No more information can be placed in the page once it is ended. The following sample code shows how to end a page in AFP Toolbox application.

**Note:** This sample shows the program looping through and placing all the pages that were created for a particular customer, as well as calculating and printing the string "Page x of y". If you do not want this calculation, all you need to issue is the End Page function call.

- **C Application**

```
for (i = 0; i < numpages, i++) {
    char buffer[15];
    currentpage = page_handles[i]
    AFPVMoveTo(currentpage,IN_2_U1440(3.875));
    AFPHMoveTo(currentpage,IN_2_U1440(7));
    sprintf(buffer," Page %d of %d",i+1, numpages);
    AFPWriteString(currentpage, buffer, -1, -1,
        LEFT_ALIGN, ' ', FALSE);
    rc = AFPEndPage(&page_handles[i]);
    checkForError(rc,"Error Ending a page/n");
}
```

- **C++ Application**

```
for (i = 0; i < numpages, i++) {
    char buffer[15];
    currentpage = page_handles[i]
    sprintf(buffer," Page %d of %d",i+1, numpages);
    text = currentpage->text();
    text->MoveTo(IN_2_U1440(7), IN_2_U1440(3.875));
    text->AddText(buffer);
    currentpage->finished();
    outputDocument.AddPage(*currentpage);
    delete currentpage;
    page_handles[i] = 0;
}
```

This example assumes that you have been building the pages one at a time and are now adding the "Page x of y" string to each page as it is ended. PAGENUM contains the current page number and PAGECOUNT is the number of pages in this set.

- **COBOL Interface**

```
03  PAGENUM          PIC 99 BINARY VALUE 1.
03  PAGENUM-OUT      PIC Z9.
03  PAGECOUNT      PIC 99 BINARY VALUE 1.
03  PAGECOUNT-OUT  PIC Z9.

...
MOVE "PAGE " TO AFP-CHARACTER-STRING
MOVE 5 TO AFP-STRING-LENGTH
MOVE FALS TO AFP-POSITION-OPTION.
MOVE LEFT-ALIGN TO AFP-ALIGNMENT-OPTION
PERFORM AFPWRITE.
MOVE PAGENUM to PAGENUM-OUT
MOVE 2 TO AFP-STRING-LENGTH
PERFORM AFPWRITE.
MOVE " OF " TO AFP-CHARACTER-STRING
MOVE 4 TO AFP-STRING-LENGTH
PERFORM AFPWRITE.
MOVE PAGECOUNT to PAGECOUNT-OUT
```

## Page Breaks

```
MOVE PAGECOUNT-OUT TO AFP-CHARACTER-STRING  
MOVE 2 TO AFP-STRING-LENGTH  
PERFORM AFPWRITE.  
PERFORM AFPEPAGE.
```

## Ending a Document

You must begin and end every document.

This call ends the document and causes the final pieces of the data stream to be written to the output destination. The following sample code shows how to end processing of AFP Toolbox application.

- **C Application**

```
rc = AFPEndDoc(&outputDocument);
```

- **C++ Application**

```
delete outputDocument;
```

- **COBOL Interface**

```
PERFORM AFPEDOC.
```

## Page Breaks

## Ending the Session

This call ends AFP Toolbox session and frees all storage. This is not done in C++.

- **C Application**

```
rc = AFPEndSession(&toolboxSession);
```

- **COBOL Interface**

```
PERFORM AFPEND.
```



## Working with Bar Codes

Bar Codes are a special type of object that might be part of a typical document. This section uses additional sample C and C++ programs to illustrate the use of bar code objects. You should already be familiar with using AFP Toolbox in order to understand the sample code shown in this section.

Figure 7 illustrates some of the terminology used in bar code objects. For detailed information about BCOCA and explanations of the parameters, refer to *Bar Code Object Content Architecture Reference*.

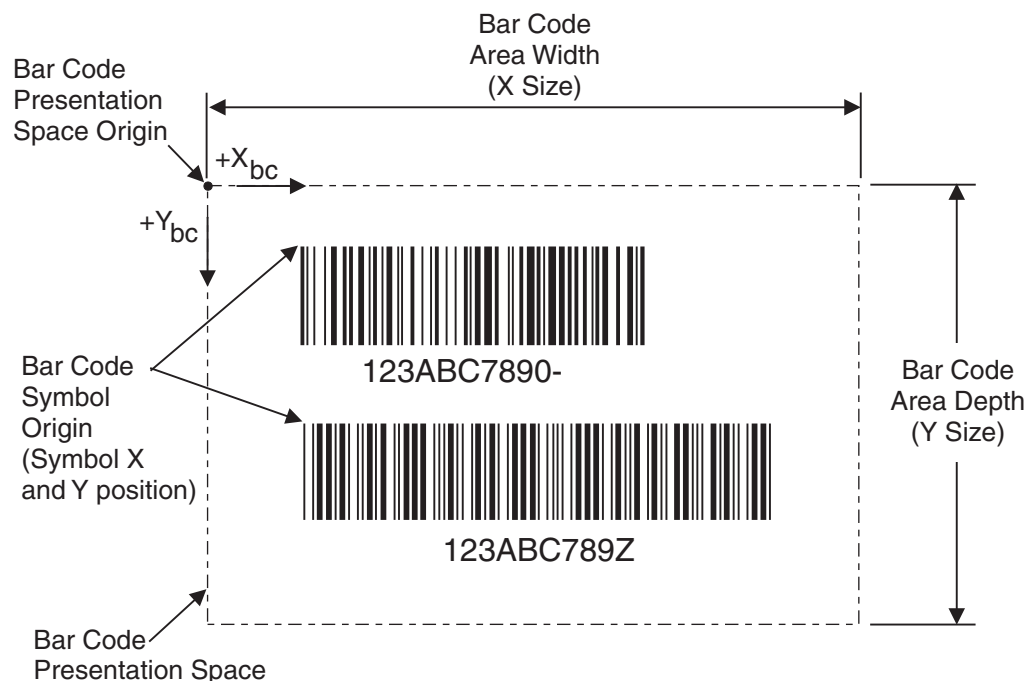


Figure 7. Bar Code Presentation Space

## Begin a Bar Code

This example defines an Interleaved 2-of-5 bar code object. The presentation space for the bar code is 120mm by 50mm, and it is rotated zero degrees relative to the page origin. A check digit is generated. The default font, modifier width, and wide-to-narrow ratio are requested, and the element height is 12mm.

### • C Application

```
TBHANDLE BarCodeHandle;          /* dcl bar code handle */
AFPBgnBarCode(PageHandle,         /* page handle */
    "BCOCA 1",                     /* name of BCOCA object */
    MM_2_U1440(120),              /* width is 120 mm */
    MM_2_U1440(50),               /* depth is 50 mm */
    DEG_0,                        /* rotation is zero */
    INTERLEAVED_2_OF_5_BC,        /* bar code type */
    0x02,                          /* modifier (chk digit) */
    DEFAULT_BCFONT,               /* default HRI font */
    BLACK,                        /* color is black */
    DEFAULT_BCMODWIDTH,           /* default modifier width */
    MM_2_U1440(12),               /* element height */
    1,                            /* element multiplier */
    -1,                           /* default ratio */
    &BarCodeHandle);
```

## Bar Codes

This example defines a Code 3 of 9 bar code object. The default presentation space for the bar code is used (size of the page) and it is rotated zero degrees relative to the page origin. No check digit is generated. Default values for HRI font, color, module width, element height, the height multiplier, and wide-to-narrow ratio are also assumed.

- **C++ Application**

```
BCOCA* bcoca = new BCOCA("Code39",
                           CODE_39_BC,
                           1); //the bar code is presented without a generated check digit
```

This example defines a PostNet bar code object. The presentation space for the bar code is 6 inches by .5 inches and it is rotated zero degrees relative to the page origin. The default font and modifier width are requested. The wide-to-narrow ratio is 1, the height multiplier is 1, and the element height is .44 inches.

- **COBOL Interface**

```
MOVE IN TO AFP-UNIT-OF-MEASURE.
PERFORM AFPSUNI.
MOVE "PostNet Object1" TO AFP-BCOCA-NAME.
MOVE 6 TO AFP-AREA-XSIZE.
MOVE .5 TO AFP-AREA-YSIZE.
MOVE DEG-0 TO AFP-OBJECT-ROTATION.
MOVE POSTNET-BC TO AFP-BC-TYPE.
MOVE 0 TO AFP-BC-MODIFIER.
MOVE DEFAULT-BCFONT TO AFP-BC-FONT-ID.
MOVE BLACK TO AFP-BC-COLOR.
MOVE DEFAULT-BCMODWIDTH TO AFP-BC-MODWIDTH.
MOVE .44 TO AFP-BC-ELHEIGHT.
MOVE 1 TO AFP-BC-HEIGHTMULT.
MOVE 1 TO AFP-BC-WIDETONARROW.
PERFORM AFPBBCD.
```

## Specifying Bar Code Data

The string "01234567890" is encoded in the output (in code page 500 since that is defined as the only valid code page for this bar code type.) The bar code symbol origin is offset five mm in the x and y directions from the bar code presentation space. HRI is requested and is to be placed on top of the printed bar code.

| **Note:** Two-dimensional bar codes have their own function calls. For more information about these  
| two-dimensional bar codes, see "Data Matrix Bar Code Data" on page 111, "MaxiCode Bar Code  
| Data" on page 175, and "PDF417 Bar Code Data" on page 186.

- **C Application**

```
rc = AFPBarCodeData(PageHandle,      /* page handle      */
                    BarCodeHandle,    /* bar code handle  */
                    "01234567890",    /* string to encode  */
                    MM_2_U1440(5),    /* x offset 5 mm    */
                    MM_2_U1440(5),    /* y offset 5 mm    */
                    TRUE,              /* format HRI       */
                    HRIOnTop,         /* place it on top  */
                    FALSE);           /* no asterisk (ignored)*/
```

The string "0123456789" is encoded in the output (in code page 500 since that is defined as the only valid code page for this bar code type). The bar code symbol origin is offset 1 inch from the left margin and 8.5 inches from the top margin of the bar code presentation space (which was defaulted to the size of the page). HRI is requested (by default) and is to be placed in the default position for this bar code type.

- **C++ Application**

```
bcoca->AddBarCode(IN_2_U1440(1),
                  IN_2_U1440(8.5),
                  "0123456789");
```

The string "65478" is encoded in the output. The bar code symbol origin is offset .1 inch from the left and top margins of the bar code presentation space.

- **COBOL Interface**

```
MOVE "65478" TO AFP-BARCODE-DATA.  
MOVE .1 TO AFP-BCSYM-XPOS.  
MOVE .1 TO AFP-BCSYM-YPOS.  
MOVE FALS TO AFP-BCHRI-PRESENT.  
MOVE HRI-DEFAULT TO AFP-BCHRI-POS.  
MOVE FALS TO AFP-BC-ASTERISK.  
PERFORM AFPBCDAT.
```

## Place Bar Code

Place the bar code into the output page.

- **C Application**

```
rc = AFPEndBarCode(PageHandle,&BarCodeHandle);
```

- **C++ Application**

```
page->IncludeBarCode(*bcoca);  
delete bcoca;
```

- **COBOL Interface**

```
PERFORM AFPEBCD.
```

### Working with Tables

Tables can present information in a logical manner. By understanding some basic principles and with some planning, you can create tables of any complexity. A table has one or more rows; each row consists of one or more columns, and each column consists of one or more subrows. AFP Toolbox constructs tables with three types of procedure calls: table, row, and field. This section defines the procedure calls and then describes how they fit together to form a table.

This is the most basic table. It has one row (horizontal), one column (vertical), one subrow, and no text:

*Table 2. Table with One Row, One Column, and No Text*

--

This is a table with one row, one subrow, two columns, and two fields with no text:

*Table 3. Table with One Row, One Subrow, Two Columns, and Two Fields with No Text*

--	--

This is a table with one row, one subrow, two columns, and two fields with text:

*Table 4. Table with One Row, One Subrow, Two Columns, and Two Fields with Text*

This is a field of text within a row.	This is another field of text in another column within a row.
---------------------------------------	---

This is a table with one row, two subrows, three columns, and five fields of text.

*Table 5. Table with One Row, Two Subrows, Three Columns, and Five Fields of Text*

This is a field of text in column 1, subrows 1 and 2.	This is a field of text in column 2, subrow 1.	This is a field of text in column 3, subrow 1.
	This is a field of text in column 2, subrow 2.	This is a field of text in column 3, subrow 2.

This is the basic approach to creating a table:

1. Define the format of the unique fields for all rows of the table using **Define Field**. This includes text positioning, text orientation, line spacing, shading, and so on. The *field* is the place where you put text.
2. Define the format of the unique rows of a table using **Define Row**. This includes arranging fields in the row, column width, minimum subrow depth, and the thickness of the top and bottom rules for the row. The *row* is the horizontal element in a table; it is divided into vertical elements called columns and can be divided horizontally into elements called subrows.
3. Begin the page using **Begin Page**.
4. Begin the table using **Begin Table**. This includes specifying table rotation, width, maximum depth, and the thickness of the rules that surround the table.
5. Begin a row using **Begin Row**. The row was defined in step 2.
6. Begin a field using **Begin Field**. The field was defined in step 2.
7. Place data in the field using **Put Text In Table Field**.
8. End the field using **End Field**.
9. Repeat steps 6-8 until all fields in the row are filled.
10. End the row using **End Row**.

11. Repeat steps 5-10 until data for the table is finished or until the end of the page is reached. See “Determining Page Breaks” on page 28 for determining when the table is full.
12. End the table using End Table.
13. End the page using End Page.

## Example

In this example, we will generate the table in Table 6. This is a simple example, but all tables are built similarly. You must define a separate Define Row call for each type of row in the table and for all the unique Define Field parameters for each field in each row. To do this, determine the number of columns and subrows in each row.

**Note:** You can use a field in more than one row, but you cannot use a field more than once in the same row.

For a complete example, see Appendix E, “AFP Toolbox Table Samples” on page 405.

Table 6. Example Table

Date	Transaction Description	Amount
------	-------------------------	--------

The following commands define the fields and rows.

### Define Field

This command defines characteristics for the field. The first time it is called, it defines characteristics for the field for row 1 column 1. It has to be called two more times to define the characteristics for the fields for row 1 column 2 and row 1 column 3, respectively. All three fields have these properties:

- No shading
- No left margin
- No right margin
- Center text in the column horizontally
- Center text in the column vertically
- No text rotation
- Table rules are .5 millimeters thick
- **For the AFP Toolbox C Library:**

```
HdrFldArray[0] = &HdrFld1
HdrFldArray[1] = &HdrFld2
HdrFldArray[2] = &HdrFld3
```

```
FormatOption = CONTER ALIGN;
VertFormat   = VERCENTER;
TxtOrient    = IOB90_T0;
AlignPos     = 0;
```

```
LeftMargin   = 0;
RightMargin  = 0;
```

```
LineSpace = 1;
```

```
ShadeIntense = 0.0;
```

```
for(Index = 0; Index < HdrFldCnt; Index++)
{
    AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
        LeftMargin, RightMargin, LineSpace, TxtOrient, ShadeIntense,
        HdrFldArray[Index]);
}
```

- **For the AFP Toolbox COBOL Interface:**

## Tables

```
MOVE FOCENTER TO AFP-FORMAT-OPTION.
MOVE VERCENTER TO AFP-VERTICAL-FORMAT.
MOVE MM TO AFP-UNIT-OF-MEASURE.
PERFORM AFPSUNI.
MOVE 0 TO AFP-SHADING-INTENSITY.
MOVE 0 TO AFP-ALIGNMENT-POSITION.
MOVE 0.0 TO AFP-LEFT-MARGIN.
MOVE 0.0 TO AFP-RIGHT-MARGIN.
MOVE AFP-DEFAULT TO AFP-LINE-SPACING.
MOVE .5 TO AFP-TOP-THICKNESS.
MOVE .5 TO AFP-BOTTOM-THICKNESS.
MOVE .5 TO AFP-LEFT-THICKNESS.
MOVE .5 TO AFP-RIGHT-THICKNESS.
PERFORM AFPDFLD.
PERFORM AFPDFLD.
MOVE AFP-FIELD-ID TO FIELDH2.
PERFORM AFPDFLD.
MOVE AFP-FIELD-ID TO FIELDH2.
```

## Define Row

This command defines characteristics for the three columns in row 1. The row has these properties:

The font used determines the subrow depth for each subrow.

Use the characteristics for row 1 column 1 that were specified in the Define Field that returned an ID of FIELDH1 (above).

Column 1 is 25 millimeters wide.

Use the characteristics for row 1 column 2 that were specified in the Define Field that returned an ID of FIELDH2 (above).

Column 2 is 70 millimeters wide.

Use the characteristics for row 1 column 3 that were specified in the Define Field that returned an ID of FIELDH3 (above).

Column 3 is 30 millimeters wide

- **For the AFP Toolbox C Library**

```
NbrofSubrows = 1;
NbrofColumns = 3;

TopThick    = MM_2_U1440(.5);
BotThick    = MM_2_U1440(.5);

ColumnWidths[0] = MM_2_U1440(25.0);
ColumnWidths[1] = MM_2_U1440(70.0);
ColumnWidths[2] = MM_2_U1440(30.0);

/* MinSubrowDepths[0] = 1.0; MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH */
MinSubrowDepths[0] = IN_2_U1440(.3);

RowArrange[0] = Hdrfld1; /* MOVE FIELD TO AFP-ROW-ARRANGE */
RowArrange[1] = Hdrfld2;
RowArrange[2] = Hdrfld3;

AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick,
  BotThick, NbrofColumns, NbrofSubrows, &RowArrange[0],
  ColumnWidths, &HdrRowId);
```

- **For the AFP Toolbox COBOL interface:**

```
MOVE MM TO AFP-UNIT-OF-MEASURE.
PERFORM AFPSUNI.
MOVE 3 TO AFP-NUMBER-COLUMNS.
MOVE 1 TO AFP-NUMBER-SUBROWS.
MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH(1).
MOVE FIELDH1 TO AFP-COLUMN-ARRANGE (1, 1).
MOVE 25.0 TO AFP-COLUMN-WIDTH (1).
MOVE FIELDH2 TO AFP-COLUMN-ARRANGE (1, 2).
```

```

MOVE 70.0 TO AFP-COLUMN-WIDTH (2).
MOVE FIELDH3 TO AFP-COLUMN-ARRANGE (1, 3).
MOVE 30.0 TO AFP-COLUMN-WIDTH (3).
PERFORM AFPDROW.

```

The following commands start the table whose maximum depth is the remaining page body space after the white space preceding the table.

### Begin Table

This command begins the table, sets up its size, and specifies the rule thickness.

- **For the AFP Toolbox C Library:**

```

RetCode = AFPBeginTable(hPage, MM_2_U1440(125.0),
    IN_2_U1440(8.0), 0, MM_2_U1440(.5), MM_2_U1440(.5),
    MM_2_U1440(.5), MM_2_U1440(.5), &hTbl);

```

- **For the AFP Toolbox COBOL interface:**

```

MOVE MM TO AFP-UNIT-OF-MEASURE.
PERFORM AFPSUNI.
COMPUTE AFP-MAX-TABLE-DEPTH = PAGE-BODY -
    TABLE-WHITE-SPACE.

MOVE 125.0 TO AFP-TABLE-WIDTH.
MOVE 1.0 TO AFP-TOP-THICKNESS.
MOVE .5 TO AFP-BOTTOM-THICKNESS.
MOVE .5 TO AFP-LEFT-THICKNESS.
MOVE .5 TO AFP-RIGHT-THICKNESS.
PERFORM AFPBTBL.

```

The following command writes the header row for the table.

### Begin Row

Begins the header row, using the ID returned from Define Row for formatting.

- **For the AFP Toolbox C Library:**

```

AFPBeginRow(hTbl, HdrRowId);

```

- **For the AFP Toolbox COBOL interface:**

```

WRITE-HEADER-ROWS.
PERFORM AFPBROW.

```

The following commands write the first field in column 1.

### Begin Field

This command begins the first field in the header row, using the ID returned from Define Field for formatting.

- **For the AFP Toolbox C Library:**

```

AFPBeginField(hTbl, Hdrfld1);

```

- **For the AFP Toolbox COBOL interface:**

```

MOVE FIELDH1 TO AFP-FIELD-ID.
PERFORM AFPBFLD.

```

### Set Font

This command sets the font for the header row.

- **For the AFP Toolbox C Library:**

```

AFPSetFont(hPage, titleFont);

```

- **For the AFP Toolbox COBOL interface:**

```

PERFORM AFPSFNT.

```

## Tables

### Trim

This command uses "Date" as the AFP-STRING-IN identifier, strips off unnecessary leading and trailing blanks in the data for proper formatting, counts the characters (length) of the data, inserts the trimmed string AFP-CHARACTER-STRING, and inserts the string length in AFP-STRING-LENGTH.

TRIM is a subprogram shipped with the example code on MVS and OS/400. It is a sample of how to trim trailing blanks from character strings. TRIM is in ATXTRIM. On OS/400 it is installed in the QAOCL library, and on MVS it is installed in the ATX.SATXSAMP data set.

**Note:** For best performance, do not call TRIM if the string is a constant length. Instead, set AFP-STRING-LENGTH and move your string directly to AFP-CHARACTER-STRING.

- **For the AFP Toolbox COBOL interface:**

```
MOVE "Date" TO AFP-STRING-IN.  
MOVE SPACES TO AFP-STRING-IN.  
MOVE "Date" TO AFP-STRING-IN.  
CALL "TRIM"  
    USING AFP-STRING-IN,  
    BY CONTENT  
    LENGTH OF AFP-STRING-IN,  
    BY REFERENCE  
    AFP-CHARACTER-STRING,  
    AFP-STRING-LENGTH.
```

### Put Text

This command puts a character string (Date) in the field and centers it.

- **For the AFP Toolbox C Library:**

```
memcpy(szTextstring, NULL, sizeof(szTextstring));  
strcpy(szTextstring, "Date");  
AFPPutTextInTableFld(hTbl, Hdrfld1,  
                    szTextstring, headingsFont);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPPCHS.
```

### End Field

This command ends the first field in row 1.

- **For the AFP Toolbox C Library:**

```
AFPEndField(hTbl);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPEFLD.
```

The following commands write the second field in column 2.

### Begin Field

This command begins the second field in the header row, using the FIELDH2 ID returned from AFPDFLD for formatting.

- **For the AFP Toolbox C Library:**

```
AFPBeginField(hTbl, Hdrfld2);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPBFLD.
```

### Put Text

This command puts a character string (Transaction Description) in the field and centers it.

- **For the AFP Toolbox C Library:**



```
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Transaction Description");
AFPPutTextInTableFld(hTbl, Hdrfld2,
                    szTextstring, headingsFont);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPPCHS.
```

### End Field

This command ends the second field in row 1.

- **For the AFP Toolbox C Library:**

```
AFPEndField(hTbl);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPEFLD.
```

The following commands write the third field in column 2.

### Begin Field

This command begins the third field in the header row, using the FIELDH3 ID returned from AFPDFLD for formatting.

- **For the AFP Toolbox C Library:**

```
AFPBeginField(hTbl, Hdrfld3);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPBFLD.
```

### Put Text

This command puts a character string (Amount) in the field and centers it.

- **For the AFP Toolbox C Library:**

```
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Amount");
AFPPutTextInTableFld(hTbl, Hdrfld3,
                    szTextstring, headingsFont);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPPCHS.
```

### End Field

This command ends the third field in row 1.

- **For the AFP Toolbox C Library:**

```
AFPEndField(hTbl);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPEFLD.
```

The following command ends row 1.

### End Row

This command ends row 1.

- **For the AFP Toolbox C Library:**

```
AFPEndRow(hTbl, HdrRowId, &CurrTableDepth);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPEROW.
```

The following command ends the table.

## Tables

### End Table

This command ends the table.

- **For the AFP Toolbox C Library:**

```
AFPEndTable(&hTbl, &CurrTableDepth);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPETBL.
```

## More About Tables

The preceding example showed the code for a row with one subrow. Here is the code and the process for developing a table that has one row with three subrows.

The parameters on the Begin Table, Define Row, and Define Field procedure calls provide flexibility in specifying different characteristics of the table. For example, you can specify shading to be used in a field. You can also specify the vertical alignment of the contents of the field. Consider the following table:

*Table 7. A Table with One Row and Three Subrows*

This field is as wide as the two fields below it.		This field extends from the top to the bottom of the row and is the only field to do so in this table.	Another field	Another field
Another field	Another field		What more can I say?	
This field is as wide as the upper, left field in this table.				

Determining the correct ARRANGE parameter values to create the previous table can be tricky. The following steps should make the process a little easier.

### Step 1. Sketch the Row

Sketch your row and assign an identifier to each field in the row. You do not have to number the fields sequentially, but the process is easier if you do.

*Table 8. Sketching the Row*

Field 1		Field 5	Field 6	Field 7
Field 2	Field 3		Field 8	
Field 4				

### Step 2. Define all of the Fields in the Row

The table has eight fields, so you must issue the Define Field call eight times and save the ID from each call. The field IDs are FIELD1, FIELD2, FIELD3, FIELD4, FIELD5, FIELD6, FIELD7, and FIELD8.

### Step 3. Form a Grid

Extend all the vertical and horizontal rules to the edges of the sketch to form a grid.

*Table 9. Forming a Grid*

Field 1	Field 1	Field 5	Field 6	Field 7
Field 2	Field 3	Field 5	Field 8	Field 8
Field 4	Field 4	Field 5	Field 8	Field 8

Extensions have been drawn to split up some fields.

## Step 4. Determine the Number of Columns and Their Widths

Determine the horizontal width of each rectangle in the grid from Step 3. These are the values to use on the Column Width parameter of Define Row.

Table 10. Determining the Number of Columns and Their Widths

12	12	50	12	12

The value of the Table Width parameter in Begin Table is the sum of the column widths, which is 98 (12 + 12 + 50 + 12 + 12).

The value of the Number Columns parameter in Define Row is 5.

## Step 5. Determine the Number Subrows and Their Depths

Determine the number of subrows in the grid from Step 3. The number of subrows is the number of horizontal divisions the row has. This is the value to use in the Number Subrows parameter in the Define Row procedure call. The value in this case is 3.

The depth of each subrow is specified in the Subrow Depth parameter in Define Row. This example uses the default subrow depth determined by the font being used.

## Step 6. Determine the Arrangement of Each Field Within Each Subrow

Construct an array called COLUMN-ARRANGE with the appropriate number of columns (in this case, five) and subrows (in this case, three), and store the arrangement of the field IDs in the elements of the array, as shown in the following figure:

Table 11. Determine the Arrangement of the Fields

Field 1	Field 1	Field 5	Field 6	Field 7
Field 2	Field 3	Field 5	Field 8	Field 8
Field 4	Field 4	Field 5	Field 8	Field 8

## Step 7. Define The Row

This Define Row call includes the IDs returned from the eight Define Field calls identified in Step 2. The example uses IDs from the eight Define Field calls but does not show the code for them. For explanation of the calls, see “Example” on page 37 or Chapter 3, “AFP Toolbox Language Reference” on page 53. The complete row definition looks like this:

- **For the AFP Toolbox C Library**

```
NbrofSubrows = 3;
NbrofColumns = 5;
```

```
TopThick    = MM_2_U1440(.5);
BotThick    = MM_2_U1440(.5);
```

```
ColumnWidths[0] = MM_2_U1440(12.0);
ColumnWidths[1] = MM_2_U1440(12.0);
ColumnWidths[2] = MM_2_U1440(50.0);
ColumnWidths[3] = MM_2_U1440(12.0);
ColumnWidths[4] = MM_2_U1440(12.0);
```

```
RowArrange[0] = FIELD1; /* MOVE FIELD TO AFP-ROW-ARRANGE */
RowArrange[1] = FIELD1;
RowArrange[2] = FIELD5;
```

## Tables

```
RowArrange[3] = FIELD6;  
RowArrange[4] = FIELD7;  
RowArrange[5] = FIELD2;  
RowArrange[6] = FIELD3;  
RowArrange[7] = FIELD5;  
RowArrange[8] = FIELD8;  
RowArrange[9] = FIELD8;  
RowArrange[10] = FIELD4;  
RowArrange[11] = FIELD4;  
RowArrange[12] = FIELD5;  
RowArrange[13] = FIELD8;  
RowArrange[14] = FIELD8;
```

```
AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick,  
    BotThick, NbrofColumns, NbrofSubrows, &RowArrange[0],  
    ColumnWidths, &HdrRowId);
```

- **For the AFP Toolbox COBOL interface:**

```
        MOVE MM TO AFP-UNIT-OF-MEASURE.  
PERFORM AFPSUNI.  
PERFORM AFPDFLD.  
    . (see the table example for the complete Define Field call)  
    .  
    .  
        MOVE 5 TO AFP-NUMBER-COLUMNS.  
        MOVE 3 TO AFP-NUMBER-SUBROWS.  
        MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH(1).  
        MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH(2).  
        MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH(3).  
        MOVE FIELD1 TO AFP-ROW-ARRANGE (1).  
        MOVE FIELD1 TO AFP-ROW-ARRANGE (2).  
        MOVE FIELD5 TO AFP-ROW-ARRANGE (3).  
        MOVE FIELD6 TO AFP-ROW-ARRANGE (4).  
        MOVE FIELD7 TO AFP-ROW-ARRANGE (5).  
        MOVE FIELD2 TO AFP-ROW-ARRANGE (6).  
        MOVE FIELD3 TO AFP-ROW-ARRANGE (7).  
        MOVE FIELD5 TO AFP-ROW-ARRANGE (8).  
        MOVE FIELD8 TO AFP-ROW-ARRANGE (9).  
        MOVE FIELD8 TO AFP-ROW-ARRANGE (10).  
        MOVE FIELD4 TO AFP-ROW-ARRANGE (11).  
        MOVE FIELD4 TO AFP-ROW-ARRANGE (12).  
        MOVE FIELD5 TO AFP-ROW-ARRANGE (13).  
        MOVE FIELD8 TO AFP-ROW-ARRANGE (14).  
        MOVE FIELD8 TO AFP-ROW-ARRANGE (15).  
        MOVE 12.0 TO AFP-COLUMN-WIDTH (1).  
        MOVE 12.0 TO AFP-COLUMN-WIDTH (2).  
        MOVE 50.0 TO AFP-COLUMN-WIDTH (3).  
        MOVE 12.0 TO AFP-COLUMN-WIDTH (4).  
        MOVE 12.0 TO AFP-COLUMN-WIDTH (5).  
PERFORM AFPDROW.
```

---

## Working with Code Pages

Code pages are used to determine which characters are printed on the page. A code page is a table that correlates graphic character IDs (hexadecimal numbers) with code points (characters). If you are entering text from a keyboard, you are using an ASCII or EBCDIC code page, depending on your operating system. When you type a particular key as input to a program, the key is translated to a hexadecimal representation by the program. The code page then maps that hexadecimal number to the appropriate character in the character set. For example, if you press a key and the program you are using translates it to X'4F', you get a different result in your output depending on the code page being used. If you are using EBCDIC code page T1V10037, you get "I". If you are using EBCDIC code page T1V10500, you get "I". If you are using EBCDIC code page T1V10259, you get the square root symbol (√). This is illustrated in Figure 8 on page 45.

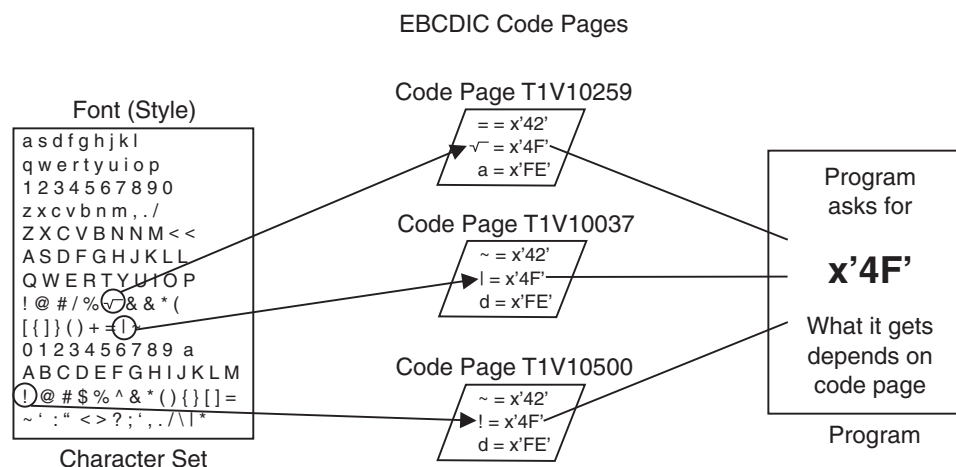


Figure 8. How Code Pages Work

By default, AFP Toolbox assumes an **input** code page based on your operating system. This code page is ASCII (T1000850) for workstation environments and EBCDIC (T1V10500) for IBM host systems. With the exception of the Measure String function, all text strings used as input to Toolbox functions are assumed to be encoded in this default code page unless it is overridden. Measure String assumes that the text is encoded in the code page associated with the current font.

AFP Toolbox translates all presentation text (text that is presented on the printed page) to the code page specified when the font for that text was defined. For example, if a Times New Roman, 10-point, bold font in code page T1000391 (Portuguese) has been defined and selected with a Set Font call, any text written with PText::AddText or Write String is translated from the current input code page to code page T1000391. The code page for bar code HRI is controlled by the bar code type and the Bar Code Architecture. For more information about bar codes, see “Bar Code Object Content Architecture (BCOCA)” on page 342.

AFP Toolbox also assumes a default **output** code page of EBCDIC T1V10500 for all non-text data stream objects, regardless of your operating system. All data stream textual values (such as the document and page names, comments, tag attributes and values, and so forth) are translated to this code page in the resulting AFP data stream unless it is overridden by Set Data Stream Codepage.

AFP Toolbox functions and methods are provided that let you override these code pages.

For more information about specifying code pages for C, C++, COBOL, and RPG programs, see the functions that set code pages for your programming language.

---

### Using Fonts with AFP Toolbox

**AIX** A font index file that matches the fonts supplied with the IBM Font Collection is shipped with AFP Toolbox. Unless you add your own font definitions, these fonts are the only ones available for use with AFP Toolbox on your system. For information about adding fonts in the AIX environment, see Appendix B, “AFP Toolbox and the AIX Environment” on page 389.

**MVS** An MVS utility named Font Library Index Program MVS (FLIPMVS) is provided with AFP Toolbox. FLIPMVS builds a font index that corresponds to the fonts installed on your MVS system. FLIPMVS should be run by your system programmer when AFP Toolbox is installed. Ask your system programmer where the font libraries and font-index data sets reside at your installation. For information about adding fonts in the MVS environment, see Appendix A, “AFP Toolbox and the OS/390 Environment” on page 375.

#### OS/400

A program is included with AFP Toolbox that can develop and diagnose problems in your output AFP documents.

In order to use AFP Toolbox on the OS/400, you must first install the AFP Font Collection (5648-B45). The command has one parameter, the name of the file where you want to display the output. If you do not specify an output file, the output goes to your terminal. For information about adding fonts in the OS/400 environment, see Appendix C, “AFP Toolbox and the AS/400 Environment” on page 395.

You can also use QYTBFLIST to verify the installation of the FontMeister component of AFP Toolbox. If FLIPLIST produces output, AFP Toolbox is installed correctly.

The output from FLIPLIST defines the valid input combinations for the Define Font By Attribute and Define Font By Name function calls in AFP Toolbox. Here is an example of some output from the FLIPLIST tool:

```
-----  
Typeface name      Style      Width      Deci-Points      Char Set      AFM File  
-----  
BOLDFACE   LATIN1   Roman      Medium      120           C08400B0      BFC  
BOOKMASTER LATIN1   Roman      Medium      50            C0B20050      EDFBL  
BOOKMASTER LATIN1   Italic     Medium      50            C0B30050      EDFBLI  
BOOKMASTER LATIN1   Roman      Medium      50            C0B40050      EDFBLB  
...  
-----
```

Refer to “Troubleshooting” on page 399 for additional information on the FLIPLIST utility.

## Printing and Viewing the Output File

To print the AFP output file, use the Print Services Facility™ (PSF) program for your environment:

- Infoprint Manager for AIX
- Infoprint Manager for Windows NT and Windows 2000
- Print Services Facility for OS/390 (PSF/390)
- Print Services Facility for OS/400 (PSF/400)

To view the AFP output file, use AFP Workbench for Windows. You can also view AFP output using the AFP Workbench Viewer component of Client Access Express on OS/400.





---

## Part 2. AFP Toolbox Language Reference

<b>Chapter 3. AFP Toolbox Language Reference</b>	53
Using the AFP Toolbox Calls to Build an AFP Document	53
Hierarchy of Calls	53
Session-level calls	54
Document-level Calls	54
Page-level Calls	55
Return Codes	58
Understanding Handles	59
Providing Positioning and Size Information	59
Format of the Function Call Descriptions	60
Including Header Files.	61
Bar Code Data	62
Begin Bar Code	66
Begin Box	73
Begin Color	75
Begin Document	78
Begin Field	82
Begin Group	84
Begin Image	86
Begin Page	91
Begin Paragraph	93
Begin Row	96
Begin Rule	97
Begin Session	100
Begin Shade	101
Begin Table	103
Begin Underscore	106
Create Link	108
I Data Matrix Bar Code Data	111
Define Double-Byte Font By Attribute	114
Define Field	117
Define Font By Attribute	120
Define Font By Attributes With Scaling	123
Define Font By Name	126
Define Row	128
Delete Font	131
Delete Page	132
End Bar Code	134
End Box	135
End Color	136
End Document	137
End Field	138
End Group	139
End Image	140
End Page	141
End Paragraph	142
End Row	143
End Rule	145
End Session	146
End Shade	147
End Table	148
End Underscore	149
Get Buffer	150

Graphic Inline . . . . .	152
Horizontal Move . . . . .	154
Horizontal Move To . . . . .	156
Image Data . . . . .	158
Image Inline . . . . .	160
Include Object . . . . .	162
Include Overlay . . . . .	167
Include Page Segment . . . . .	169
Invoke Medium Map . . . . .	171
Keep Object . . . . .	173
I MaxiCode Bar Code Data . . . . .	175
Measure Double-Byte String . . . . .	178
Measure String . . . . .	180
Next Line . . . . .	183
Output Comment . . . . .	184
I PDF417 Bar Code Data . . . . .	186
Preload Object . . . . .	189
Preload Overlay . . . . .	191
Put Box . . . . .	192
Put Color . . . . .	194
Put Double-Byte Text. . . . .	196
Put Horizontal Rule . . . . .	198
Put Shade . . . . .	200
Put Tag. . . . .	201
Put Text . . . . .	203
Put Text in Table Field . . . . .	205
Put Vertical Rule . . . . .	207
Query . . . . .	209
Repeat String . . . . .	211
Rotate Overlay . . . . .	213
Set Color . . . . .	215
Set Data Stream Codepage . . . . .	217
Set Extended Color . . . . .	219
Set Font . . . . .	221
Set Font Type . . . . .	223
Set Input Codepage . . . . .	225
Set Input Double-Byte Codeset . . . . .	228
Set Media Size . . . . .	230
Set Object Color Profile. . . . .	232
Set Position . . . . .	234
Set Text Codepage . . . . .	236
Set Text Double-Byte Codeset . . . . .	238
Set Text Orientation . . . . .	240
Set Units . . . . .	242
Translate And Measure Double-Byte String . . . . .	244
Translate And Measure String . . . . .	246
Vertical Move . . . . .	249
Vertical Move To . . . . .	251
Write Double-Byte String . . . . .	253
Write String . . . . .	256
Font Return Codes . . . . .	259
 <b>Chapter 4. AFP Toolbox GOCA Functions . . . . .</b>	 261
Graphic Presentation Space Concepts . . . . .	261
C++ Objects . . . . .	261
C and COBOL Function Descriptions . . . . .	261

Header files and Copy Books . . . . .	262
Begin Graphic Object . . . . .	263
End Graphic Object . . . . .	266
GOCA Begin Area . . . . .	267
GOCA Character String . . . . .	269
GOCA Circle . . . . .	271
GOCA Draw Box . . . . .	272
GOCA Draw Objects . . . . .	274
GOCA Ellipse . . . . .	277
GOCA End Area . . . . .	280
GOCA No Operation . . . . .	281
GOCA Partial Circle . . . . .	282
GOCA Partial Ellipse . . . . .	284
GOCA Query GPS Position . . . . .	286
GOCA Set Character Parameters . . . . .	287
GOCA Set Font . . . . .	290
GOCA Set GPS Position . . . . .	291
GOCA Set Extended Color . . . . .	292
GOCA Set Line Parameters . . . . .	294
GOCA Set Marker Parameters . . . . .	296
GOCA Set Pattern Symbol . . . . .	298
GOCA Set Process Color . . . . .	301



---

## Chapter 3. AFP Toolbox Language Reference

This section describes the functions provided with the AFP Toolbox for use by the C, COBOL, and RPG language programs. To use these functions, you need to be aware of the structure of AFP documents but do not need to understand the semantics or syntax of the MO:DCA data stream.

The sections that follow describe the supported objects and show the syntax of the function calls used to generate documents and pages containing these objects.

---

### Using the AFP Toolbox Calls to Build an AFP Document

A typical sequence of calls for an application using the AFP Toolbox functions is:

1. Initialize the AFP application environment by calling Begin Session. The Begin Session call must complete successfully before you issue any other AFP Toolbox function calls.
2. Call Begin Document.
3. Before placing text, define any fonts to use in this document. A font only needs to be defined once as it is being used and need not be redefined unless another font has been defined and you want to use the previous font again.
4. Call Begin Page.
5. Specify where to put the next data on the page by calling Set Position, Horizontal Move, Horizontal Move To, Vertical Move, Vertical Move To, or Next Line. You can use Measure String and Query as needed to calculate the new position for data.
6. Specify any changes to the attributes of the data by calling Set Font, Set Font Type, and Set Color. Do not call these functions if the value of the attribute has not changed since last time you placed data.
7. Place the data using the appropriate call:
  - a. Write String and Write Double-Byte String to place a single line of text
  - b. Begin Paragraph, End Paragraph, Put Text and Put Double-Byte Text to flow strings of text into a paragraph
  - c. Put Horizontal Rule, Put Vertical Rule, Begin Rule, and End Rule to draw horizontal or vertical rules
  - d. Put Box, Begin Box, and End Box to draw boxes
  - e. Graphic Inline, Image Inline, Begin Bar Code, End Bar Code, Include Object, Include Overlay, and Include Page Segment to include an external resource or other object in the output
  - f. Repeat String to repeat a string of characters
  - g. Output Comment to place a comment into this page
  - h. Create Link or Put Tag to insert indexing information
8. Repeat steps 5 through 7 until the page is finished.
9. Call End Page.
10. If buffered output was specified on Begin Document, issue Get Buffer calls repeatedly until there is no more data to receive for this page.
11. Repeat step 4 through 10 until all pages are finished.
12. Call End Document
13. Call End Session

---

### Hierarchy of Calls

AFP Toolbox has three levels of program calls:

#### Session-level calls

These calls begin and end an AFP Toolbox session, set overall session parameters, and are issued only once for each application.

## **Document-level calls**

These calls begin and end an AFP Toolbox document, set overall document parameters, and place data at the document level.

## **Page-level calls.**

These calls begin and end an AFP Toolbox page, set overall page parameters, and place data within individual pages.

## **Session-level calls**

### **Begin Session**

Starts the AFP Toolbox session.

### **Set Data Stream Codepage**

Specifies the code page name used for encoding character strings in the output data stream.

### **Set Input Codepage**

Specifies the code page name that is used for encoding all input character strings for the AFP Toolbox session.

### **Set Input Double-Byte Codeset**

Specifies the code set name that is used for encoding text strings of DBCS.

### **End Session**

Ends the AFP Toolbox session.

## **Document-level Calls**

### **Begin Document**

Begins the AFP Toolbox document and specifies the name used in the output data stream.

### **Invoke Medium Map**

Builds a request in the output to switch copy groups in the current form definition.

### **Output Comment**

Lets you put comments in the data stream for use with postprocessors or other user-defined applications.

### **Put Tag**

Creates an indexing tag in the document for use by presentation systems or postprocessor applications.

### **Begin Group**

Begins a logical group of pages.

### **End Group**

Ends a previously started logical group of pages.

### **Set Media Size**

Sets the size of the media (printable area) for subsequent pages in the document.

### **Define Field**

Creates a field definition for a table.

### **| Define Font By Attribute**

| Finds a font on your system that matches specified attributes and returns a font ID corresponding  
| to this font

### **| Define Font By Attribute With Scaling**

| Finds a font on your system that matches specified attributes and returns a font ID corresponding  
| to this font.

| **Define Font By Name**  
| Finds a font on your system that matches the specified character set and code page combination  
| and returns a font ID.

**Define Row**  
Creates a row definition for use in a table.

**End Document**  
Ends the AFP Toolbox document and causes the remainder of the data stream to be written to the output destination.

**Delete Font**  
Deletes the specified font.

**Preload Object**  
Specifies that the object is to be sent to the printer before printing starts.

**Preload Overlay**  
Specifies that the overlay is to be sent to the printer before printing starts.

## Page-level Calls

**Begin Page**  
Begins and names a logical page.

**End Page**  
Ends the logical page.

**Begin Bar Code**  
Provides information about the bar code generated.

**Bar Code Data**  
Adds data to the bar code previously started with Begin Bar Code.

| **Data Matrix Bar Code Data**  
| Adds data to Data Matrix bar code previously started with Begin Bar Code.

| **MaxiCode Bar Code Data**  
| Adds data to a MaxiCode bar code previously started with Begin Bar Code.

| **PDF417 Bar Code Data**  
| Adds data to a PDF417 bar code previously started with Begin Bar Code.

**End Bar Code**  
Ends the bar code that was started with Begin Bar Code.

**Put Box**  
Draws a box from the current position.

**Begin Box**  
Begins the box at the current location.

**End Box**  
Ends the box at the current location.

**Begin Graphic Object**  
Begins a graphic object in the specified page.

**End Graphic Object**  
Ends a graphic object in the specified page.

**Begin Color**  
Begins a color at the current location.

**Put Color**  
Places a color at the current location.

**End Color**

Ends a color at the current location.

**Graphic Inline**

Reads a previously created GOCA object and includes it inline in the data stream at the current position.

**Image Inline**

Reads a previously created IOCA or IM1 image and includes it inline in the data stream at the current position.

**Begin Image**

Begins building an image object at the current position.

**Image Data**

Include image data records into an image that was started with Begin Image.

**End Image**

Ends the image object previously started with the corresponding Begin Image.

**Begin Paragraph**

Begins a paragraph.

**Put Text**

Adds a string of text to the current paragraph.

**| Measure String**

| Measures the width of the specified string in the given font and returns the width.

**| Translate and Measure Double-Byte String**

| Translates the string DBCS characters from **in\_inputCodeSet** to the code set requested when  
| **in\_fontID** font was defined using AFP Define Double-Byte Font By Attribute, then measures the  
| translated string using the metrics from the specified font.

**| Translate and Measure String**

| Translates the string of characters from **in\_inputCodePage** to the code page requested when  
| **in\_fontID** font was defined, then measures the translated string using the metrics from the  
| specified font.

**| Define Double-Byte Font by Attribute**

| Finds a font on your system that matches specified character set and code page combinations and  
| returns a font ID.

**Put Double-Byte Text**

Adds a string of DBCS text to the current paragraph.

**| Measure Double-Byte String**

| Measures the width of the specified string text in the given font and returns the width.

**End Paragraph**

Ends the paragraph previously started with Begin Paragraph.

**Begin Shade**

Begins shading at the current location.

**Put Shade**

Puts a shaded rectangle at the current location.

**End Shade**

Ends shading at the current location.

**Put Horizontal Rule**

Draws a horizontal rule from the current position.



**Put Vertical Rule**

Draws a vertical rule from the current position.

**Begin Rule**

Begins drawing a solid rule in a horizontal or vertical direction from the current position.

**End Rule**

Ends the horizontal or vertical rule previously started with Begin Rule.

**Create Link**

Creates a link area at the current location to a specified target.

**Get Buffer**

Returns the next buffer of output for the page.

**Horizontal Move**

Move inline horizontally relative to the current inline coordinate position.

**Horizontal Move To**

Move inline horizontally to a specified position.

**Vertical Move**

Move vertically, that is, move the baseline position, relative to the current position.

**Vertical Move To**

Move vertically, that is, move the baseline position, to a specified position.

**Include Object**

Includes a reference to an image, graphic, bar code, or non-AFP object at the current position.

**Include Overlay**

Creates a reference to an overlay at the current position.

**Rotate Overlay**

Includes a page overlay at the current location and gives a rotation value.

**Include Page Segment**

Creates a reference to a page segment at the current position.

**Next Line**

Advances one line in the current font.

**Query** Returns the current values for position, color, and font information.

**Repeat String**

Repeat a string of characters at the current position.

**Set Color**

Specifies the color for subsequent data (text and rules).

**Set Extended Color**

The Set Extended Color control sequence specifies a color value and defines the color specified encoding for that value.

**Set Font**

Specifies the font for subsequent text data.

**Set Font Type**

Specifies the font technology for subsequent font definitions.

**Set Text Codepage**

Specifies the code page name that is used for encoding **only input** text character strings.

**Set Text Double-Byte Codeset**

Specifies the codeset name that is used for encoding **input** text character strings of DBCS.

**Set Text Orientation**

Moves the page origin including the coordinate system for all subsequent PText operations.

**Set Position**

Sets the new position for output on the page.

**Set Units**

Sets the units of measure for most following calls. (COBOL only)

**Write Double-Byte String**

Places DBCS text into the current page at the current position.

**Write String**

Places text into the current page at the current position.

**Begin Table**

Begins a table at the current position.

**End Table**

Ends a table

**Begin Row**

Begins a new row in a table.

**End Row**

Ends a row in a table.

**Begin Field**

Begins a field of data in a row of a table.

**End Field**

Ends the field in a row of a table.

**Delete Page**

Deletes a page.

**Begin Underscore**

Begins an underscore.

**End Underscore**

Ends an underscore.

**Put Text in Table Field**

Puts text in the current field in a table.

**Keep Object**

Specifies that the object is to be kept at the printer while the page is printing.

**Set Object Color Profile**

Identifies the color profile to use when printing an object.

---

**Return Codes**

Each function returns an integer value as the return code. Normal processing results in a returned value of 0. If a non-zero value is returned, some type of error has occurred. The various return codes possible for each function are described with the syntax diagram for that function.

**Note:** To set the system return codes in C programs on MVS, your application must return an integer from main.

---

## Understanding Handles

Handles are addresses that identify the portion of the document or page that you are currently processing in your application. The AFP Toolbox uses these handles to keep track of which “state” in the document it is processing at any given time. For example, the page handle on an **WriteString** call identifies the page in which the text string is placed (since you can write to more than one page at a time).

In general, you don’t need to worry much about handles, because they are returned on the function calls and your application should not change them. The exceptions are:

1. If you are building more than one page or document at a time, you must ensure that the page handle and document handle variables always contain the correct value.
2. If you are creating multiple rules or boxes, you need to keep track of the various handles in order to end the correct rule or box.

---

## Providing Positioning and Size Information

### For the AFP Toolbox C Library:

The input to all functions for positioning and size parameters is always in units of 1440th of an inch. If you want to use other units of measure such as millimeters or inches, you can use the following functions to convert values:

```
IN_2_U1440(x)      /* converts from inches to 1440ths */
MM_2_U1440(x)      /* converts from millimeters to 1440ths */
CM_2_U1440(x)      /* converts from centimeters to 1440ths */
U240_2_U1440(x)    /* converts from 240 pels per inch to 1440ths */
```

```
/* convert from 1440ths to inches, mm, cm, 240ths */
```

```
U1440_2_IN(x)
U1440_2_MM(x)
U1440_2_CM(x)
U1440_2_U240(x)
U1440_2_U72000(x)
```

### For the AFP Toolbox COBOL interface:

The input to all functions for positioning and sizing parameters is always in units of 1440ths per inch (unless specified otherwise on a particular parameter). If you want to use other units of measure such as inches or millimeters, you can use the ATXUNITS procedure to convert values.

ATXUNITS is supplied in the ATX.SATXSAMP data set.<sup>2</sup>Use the COBOL copy statement to copy it into your program. It takes three input values:

```
PIC XX indicating unit of measure for input value
    IN = inches
    MM = millimeters
    CM = centimeters
    PL = 240 pels per inch
PIC S9(4)V9(3) field containing the input value
PIC S9(4)V9(3) field to receive the converted value
```

ATXUNITS returns the converted value in the field you specify, and sets the COBOL variable RETURN-CODE to 0 if successful and 8 if the conversion failed.

To use ATXUNITS to convert a centimeter value to 1440ths of an inch:

---

2. If you cannot find this data set, contact your system administrator

```
MOVE "cm" TO MY-UOM.  
MOVE 2.54 TO MY-ORIGINAL-VALUE.  
CALL "ATXUNITS" using MY-UOM,  
MY-ORIGINAL-VALUE,  
MY-NEW-VALUE.
```

Data item MY-NEW-VALUE contains the converted value.

**Note:** When converting from one unit of measurement to another, rounding factors might cause some differences in your overall dimensions. To provide device-independent formatting, the output data stream produced by AFP Toolbox has a resolution of 1440 ppi (pels per inch). The printer rounds positioning information from 1440ths of an inch to the nearest pel (240, 300, and so forth). This might cause one pel rounding errors that could be evident in rule widths and rule intersections.

APAR PQ30890 introduces an alternative to ATXUNITS for COBOL. This APAR lets you do a PERFORM of AFPSUNI to set the units for the document to be inches or millimeters. AFPSUNI uses the COBOL Set Units function. COBOL Set Units has better performance and is easier to use than ATXUNITS. See “Set Units” on page 242 for more information.

---

## Format of the Function Call Descriptions

The function descriptions are listed in alphabetic order. The actual sequence for issuing calls should be as described in “Using the AFP Toolbox Calls to Build an AFP Document” on page 53.

Each function call description includes the following sections:

### Function

A description of the major purpose of each function.

### Syntax

A diagram showing the function parameters.

### Input Parameters

Explanation of each input parameter. In the syntax diagrams for each function call, the input parameters start with `in_`.

### Output Parameters

Explanation of each output parameter. In the syntax diagrams for each function call, the output parameters start with `outp_`.

### Return Codes

List of possible return codes (other than zero, which is normal).

### Sample Function Call

Provides sample code for using the function. All sample functions assume that prerequisite calls and variable definitions have been made prior to the call. In other words, only the actual function is shown and not everything that would need to be coded before the function call.

When calling these functions, *every parameter must be specified and given in the order shown in the diagram.*

The following data types are used in the syntax diagrams to indicate the type of parameters:

### HANDLE

An identifier for a session, document, page, or other type of object structure.

### BOOLEAN

An integer with a value of 0 (FALSE) or non-zero (TRUE).

### FontID

For choosing a font when formatting text.

---

## Including Header Files

In order to use the functions described in this chapter, you must include the AFP Toolbox header file for your operating system once in your application program. See the appropriate appendix for your operating system:

- Appendix A, “AFP Toolbox and the OS/390 Environment” on page 375.
- Appendix B, “AFP Toolbox and the AIX Environment” on page 389.
- Appendix C, “AFP Toolbox and the AS/400 Environment” on page 395.

---

## Bar Code Data

### Function

Adds data to a bar code previously started with Begin Bar Code, and specifies whether and how the HRI should be presented. The bar code must be placed such that the entire bar code, including any HRI, fits within the defined presentation space (see Figure 7 on page 33) and also fits on the page. If not, an error will occur when it is printed.

When creating bar code objects, you should be familiar with the standard bar code programming techniques and values. If you set incorrect values for a bar code type, you get errors when it prints. Refer to *Bar Code Object Content Architecture Reference* for more information about creating bar codes. Invalid data or invalid bar code symbology values result in errors when your document is printed.

For UPS and EAN bar codes, the bar code data is translated to code page T1000893. For the remaining bar code types, the bar code data is translated to code page T1V10500. Code pages T1000893 and T1V10500 must exist in the font library available to the AFP Toolbox.

- | Two-dimensional bar codes have their own function calls. For more information about these
- | two-dimensional bar codes, see “Data Matrix Bar Code Data” on page 111, “MaxiCode Bar Code Data” on
- | page 175, and “PDF417 Bar Code Data” on page 186.

### Syntax

#### For the AFP Toolbox C Library:

```
AFPBarCodeData(  
    TBHANDLE    in_pageHandle,  
    TBHANDLE    in_bcHandle,  
    char*       in_bcdData,  
    ushort      in_bcsymXpos,  
    ushort      in_bcsymYpos,  
    boolean     in_presentHRI,  
    ushort      in_HRIposition,  
    boolean     in_asterisk  
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPBCDAT.  
    CALL "CBLBarCodeData"  
        USING  
            BY VALUE  
                AFP-PAGE-HANDLE  
                AFP-BARCODE-HANDLE  
            BY CONTENT  
                AFP-BARCODE-DATA  
            BY VALUE  
                AFP-BCSYM-XPOS  
                AFP-BCSYM-YPOS  
                AFP-BCHRI-PRESENT  
                AFP-BCHRI-POS  
                AFP-BC-ASTERISK  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLBarCodeData" TO AFP-ERRDATA.  
    PERFORM CHKSUC.
```

#### For the AFP Toolbox RPG Interface:

```

DAFPBCDAT      PR      10I 0 EXTPROC('AFPBarCodeData')
D  AFPPAGHDL      *      VALUE
D  AFPBCHDL      *      VALUE
D  AFPBCDATA      50
D  AFPBCXPOS      10I 0 VALUE
D  AFPBCYPOS      10I 0 VALUE
D  AFPBCHRIP      10I 0 VALUE
D  AFPBCHRPS      5I 0 VALUE
D  AFPBCASTR      10I 0 VALUE

```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the BgnPage call.

### Bar code handle

The handle for the bar code. Bar code handle is returned by the Begin Bar Code call.

### Data

The actual data to encode, specified as a series of single-byte code points from a specific code page. The assumed code page and the valid code points for the data depend on the type of bar code being generated, and are described in *Bar Code Object Content Architecture Reference*.

The output code page for the bar code data depends on the bar code type, and is EBCDIC code page T1V10500 or T1000893. The input code page is controlled with the SetInputCodepage function (or the default).

### symbol x position

Specifies the X offset of the top left corner of the left most element of the bar code symbol from the presentation space origin. The valid range for the position is 1 to 32767 1440ths. Zero is not valid.

### symbol y position

Specifies the Y offset of the top left corner of the left most element of the bar code symbol from the presentation space origin. The valid range for the position is 1 to 32767 1440ths. Zero is not valid. If you specify that HRI should be presented on top of the bar code, the offset position must allow enough room the text.

### Present HRI

Specifies whether the HRI of the bar code data should be printed. Some bar code types ignore the HRI request. Values are **true** and **false**.

### HRI Position

Specifies the location of the HRI if Present HRI is TRUE. Valid values are:

#### For the AFP Toolbox C Library:

```

HRIDefault = 0x00,    // presentation device default
HRIOnBottom = 0x01,   // place HRI below the symbol
HRIOnTop = 0x02       // place HRI above the symbol

```

#### For the AFP Toolbox COBOL Interface:

```

HRI-DEFAULT    // presentation device default
HRI-ON-BOTTOM  // place HRI below the symbol
HRI-ON-TOP     // place HRI above the symbol

```

#### For the AFP Toolbox RPG Interface:

```

HRIDEFAULT    // presentation device default
HRIONBOT      // place HRI below the symbol
HRIONTOP      // place HRI above the symbol

```

## Bar Code Data

### Asterisk

Specifies whether an asterisk should be presented as the HRI for Code 39 bar code start and stop characters. This value is ignored for other bar code types. Possible values are **true** and **false**. The default is **false**.

## Return Codes

- 1 The bar code handle value is null. You must specify a valid address for this parameter.
- 4 Page handle is not pointing to a valid page. Ensure you have called Begin Page before issuing this call.
- 17 Bar code handle is not pointing to a valid bar code object. Ensure you have called Begin Bar Code before issuing this call.

## Sample Function Call

### • For the AFP Toolbox C Library:

This example shows an Interleaved 2-of-5 bar code, generates a check digit (modifier of X'02'), uses the default font for the HRI (which is generated & placed above the bar code).

```
rc = AFPBgnBarCode(PageHandle,"BCOCA 1",MM_2_U1440(120),MM_2_U1440(50),
    DEG_0,INTERLEAVED_2_OF_5_BC,0x02,DEFAULT_BCFONT,
    BLACK,DEFAULT_BCMODWIDTH,MM_2_U1440(12),
    1,-1,&BarCodeHandle);
```

```
rc = AFPBarCodeData(PageHandle,BarCodeHandle,"01234567890",
    MM_2_U1440(5),MM_2_U1440(5),
    TRUE,HRIOnTop,FALSE);
```

```
rc = AFPEndBarCode(PageHandle,&BarCodeHandle);
```

### • For the AFP Toolbox COBOL Interface:

This example shows a UPCA barcode with HRI below the barcode.

```
MOVE "UPCA Object2" TO AFP-BCOCA-NAME.
MOVE 6803           TO AFP-AREA-XSIZE.
MOVE 2834           TO AFP-AREA-YSIZE.
MOVE DEG-0          TO AFP-OBJECT-ROTATION.
MOVE UPC-A-BC       TO AFP-BC-TYPE.
MOVE 0              TO AFP-BC-MODIFIER.
MOVE DEFAULT-BCFONT TO AFP-BC-FONT-ID.
MOVE BLACK          TO AFP-BC-COLOR.
MOVE DEFAULT-BCMODWIDTH TO AFP-BC-MODWIDTH.
MOVE 680            TO AFP-BC-ELHEIGHT.
MOVE 1              TO AFP-BC-HEIGHTMULT.
MOVE 1              TO AFP-BC-WIDETONARROW.
PERFORM AFPBBCD.
MOVE "01234567890" TO AFP-BARCODE-DATA.
MOVE 283            TO AFP-BCSYM-XPOS.
MOVE 283            TO AFP-BCSYM-YPOS.
MOVE TRU            TO AFP-BCHRI-PRESENT.
MOVE HRI-ON-BOTTOM TO AFP-BCHRI-POS.
MOVE TRU            TO AFP-BC-ASTERISK.
PERFORM AFPBCDAT.
PERFORM AFPEBCD.
```

### • For the AFP Toolbox RPG Interface

\*Provide general info about the bar code generated.

C	MOVE	'BCOCA1'	AFBPCNAM
C	CAT(P)	X'00':0	AFBPCNAM
C	Z-ADD	7200	AFBPCXSIZ
C	Z-ADD	2880	AFBPCYSIZ
C	Z-ADD	DEG0	AFBPCRTAT
C	Z-ADD	INTRL20F5	AFPBCTYPE
C	MOVE	x'02'	AFPBCTMOD



## Bar Code Data

C	Z-ADD	1	AFBFCFONT
C	Z-ADD	BLACK	AFPBCCOL
C	Z-ADD	-1	AFPBMODW
C	Z-ADD	-1	AFPBCELHT
C	Z-ADD	1	AFPBCHMUL
C	Z-ADD	-1	AFPBW2N
C	EXSR	AFPBBCDR	
*Add data to above bar code and specify how the HRI should be presented.			
C	MOVE	'12345678'	AFPBCDATA
C	CAT(P)	X'00':0	AFPBCDATA
C	Z-ADD	1	AFPBXPOS
C	Z-ADD	1	AFBCYPOS
C	MOVE	TRU	AFPBCHRIP
C	MOVE	HRIONBOT	AFPBCHRPS
C	Z-ADD	FALS	AFPBCASTR

---

# Begin Bar Code

## Function

Provide general information about the bar code generated. The actual bar code data is specified with the Bar Code Data function call. Do not issue this call within a paragraph or a table.

## Syntax

### For the AFP Toolbox C Library:

```
AFPBgnBarCode(  
    TBHANDLE      in_pageHandle,  
    char*         in_BCOCAName,  
    ulong         in_areaxsize,  
    ulong         in_areaysize,  
    DegRot        in_objrotation,  
    BarCodeType   in_bcType,  
    ushort        in_bcMod,  
    FontID        in_briFont,  
    MODCColors    in_bcColor,  
    ushort        in_modWidth,  
    ushort        in_elHeight,  
    ushort        in_heightMult,  
    ushort        inwideToNarrow,  
    TBHANDLE*     outp_bcHandle  
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPBBCD.  
    CALL "CBLBgnBarCode"  
        USING  
            BY VALUE  
                AFP-PAGE-HANDLE  
            BY CONTENT  
                AFP-BCOCA-NAME  
            BY VALUE  
                AFP-AREA-XSIZE  
                AFP-AREA-YSIZE  
                AFP-OBJECT-ROTATION  
                AFP-BC-TYPE  
                AFP-BC-MODIFIER  
                AFP-BC-FONT-ID  
                AFP-BC-COLOR  
                AFP-BC-MODWIDTH  
                AFP-BC-ELHEIGHT  
                AFP-BC-HEIGHTMULT  
                AFP-BC-WIDETONARROW  
            BY REFERENCE  
                AFP-BARCODE-HANDLE  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLBgnBarCode" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```

DAFPBBBD      PR      10I 0 EXTPROC('AFPBgnBarCode')
D  AFPPAGHDL      *    VALUE
D  AFPBCNAM      250
D  AFPBCXSIZ      10I 0 VALUE
D  AFPBCYSIZ      10I 0 VALUE
D  AFPBCRTAT      10I 0 VALUE
D  AFPBCTYPE      10I 0 VALUE
D  AFPBCMOD      5I 0 VALUE
D  AFPBCFONT      5I 0 VALUE
D  AFPBCCOL      10I 0 VALUE
D  AFPBCMODW      5I 0 VALUE
D  AFPBCELHT      5I 0 VALUE
D  AFPBCHMUL      5I 0 VALUE
D  AFPBCW2N      5I 0 VALUE
D  AFPBCHDL      *

```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### BCOCA name

Up to 252 characters to use as the name of this object in the output data stream. The first eight characters are used in the Begin Bar Code structured field as the bar code data object name. If more than eight characters are specified, an FQN triplet will also be built to contain the entire string.

### Area size

These parameters specify the size of the area (presentation space) for the bar code data (see Figure 7 on page 33). The area x size and area y size are the width and depth of the space respectively.

### Object Rotation

Specifies the clockwise rotation of the object area from the x axis rotation of the page. Valid values are:

#### **DEG\_0 (for C), DEG-0 (for COBOL), or DEG0 (for RPG)**

The object area is not rotated.

#### **DEG\_90 (for C), DEG-90 (for COBOL), or DEG90 (for RPG)**

The object area is rotated 90 degrees clockwise.

#### **DEG\_180 (for C), DEG-180 (for COBOL), or DEG180 (for RPG)**

The object area is rotated 180 degrees clockwise.

#### **DEG\_270 (for C), DEG-270 (for COBOL), or DEG270 (for RPG)**

The object area is rotated 270 degrees clockwise.

### Type

The type of bar code symbol generated. Many subsequent parameters are dependent on the bar code type, so be sure you understand how your type of bar codes should be implemented.<sup>3</sup> Valid values are:

3. The abbreviations used in these descriptions have the following meanings:

AIM USS: Automatic Identification Manufacturers Uniform Symbol Specification

EAN: European Article Numbering

JAN: Japanese Article Numbering

MSI: MSI Data Corporation

POSTNET: POSTal Numeric Encoding Technique (U.S. Postal Service)

UPC: Universal Product Code (United States)

UPC/CGPC: Universal Product Code (United States) and the Canadian Grocery Product Code

## Begin Bar Code

### For the AFP Toolbox C Library:

```
CODE_39_BC or AIM_USS_39_BC // AIM USS-39, Code 39 (3 of 9)
MSI_BC // Modified Plessey
UPC_A_BC or UPC_CGPC_VER_A_BC // UPC/CGPC Version A
UPC_E_BC or UPC_CGPC_VER_E_BC // UPC/CGPC Version B
UPC_2_DIGIT_SUPP_BC // UPC - two digit supplemental
UPC_5_DIGIT_SUPP_BC // UPC - five digit supplemental
EAN_8_BC // EAN-8 (includes JAN-short)
EAN_13_BC // EAN-13 (includes JAN-standard)
INDUSTRIAL_2_OF_5_BC // Industrial 2-of-5
MATRIX_2_OF_5_BC // Matrix 2-of-5
AIM_USS_25_BC or INTERLEAVED_2_OF_5_BC // Interleaved 2-of-5, AIM USS-I 2/5
CODABAR_BC or CODABAR_2_OF_7_BC or AIM_USS_CODABAR_BC // AIM USS-Codabar, Codabar 2-of-7
CODE_128_BC or AIM_USS_128_BC // CODE 128, AIM USS-128
EAN_2_DIGIT_SUPP_BC // EAN Two-digit supplemental
EAN_5_DIGIT_SUPP_BC // EAN Five-digit supplemental
POSTNET_BC // POSTNET
ROYAL_MAIL_BC // postal bar code for the UK
JAPAN_POST_BC // postal bar code for Japan
DMATRIX_BC // data matrix
MAXICODE_BC // maxicode
PDF417_BC // PDF417
AUSTRALIA_POST_BC // postal bar code for Australia
```

### For the AFP Toolbox COBOL Interface:

```
CODE-39-BC or AIM-USS-39-BC // AIM USS-39, Code 39 (3 of 9)
MSI-BC // Modified Plessey
UPC-A-BC or UPC-CGPC-VER-A-BC // UPC/CGPC Version A
UPC-E-BC or UPC-CGPC-VER-E-BC // UPC/CGPC Version B
UPC-2-DIGIT-SUPP-BC // UPC - two digit supplemental
UPC-5-DIGIT-SUPP-BC // UPC - five digit supplemental
EAN-8-BC // EAN-8 (includes JAN-short)
EAN-13-BC // EAN-13 (includes JAN-standard)
INDUSTRIAL-2-OF-5-BC // Industrial 2-of-5
MATRIX-2-OF-5-BC // Matrix 2-of-5
AIM-USS-25-BC or INTERLEAVED-2-OF-5-BC // Interleaved 2-of-5, AIM USS-I 2/5
CODABAR-BC or CODABAR-2-OF-7-BC or AIM-USS-CODABAR-BC // AIM USS-Codabar, Codabar 2-of-7
CODE-128-BC or AIM-USS-128-BC // CODE 128, AIM USS-128
EAN-2-DIGIT-SUPP-BC // EAN Two-digit supplemental
EAN-5-DIGIT-SUPP-BC // EAN Five-digit supplemental
POSTNET-BC // POSTNET
ROYAL-MAIL-BC // postal bar code for the UK
JAPAN-POST-BC // postal bar code for Japan
DMATRIX-BC // data matrix
MAXICODE-BC // maxicode
PDF417-BC // PDF417
AUSTRALIA_POST_BC // postal bar code for Australia
```

### For the AFP Toolbox RPG Interface:

```
CODE39 or AIMUSS39 // AIM USS-39, Code 39 (3 of 9)
MSI // Modified Plessey
UPCA // UPC/CGPC Version A
UPCB // UPC/CGPC Version B
UPC2DIGSP // UPC - two digit supplemental
UPC5DIGSP // UPC - five digit supplemental
EAN8 // EAN-8 (includes JAN-short)
EAN13 // EAN-13 (includes JAN-standard)
IND2OF5 // Industrial 2-of-5
MATRX2OF5 // Matrix 2-of-5
```

AIMUSS25 or INTRL20F5	// Interleaved 2-of-5, AIM USS-I 2/5
CODABAR or CODABAR27	// AIM USS-Codabar, Codabar 2-of-7
CODE128 or AIMUSS128	// CODE 128, AIM USS-128
EAN2DIGIT	// EAN Two-digit supplemental
EAN5DIGIT	// EAN Five-digit supplemental
POSTNET	

**Modifier**

The modifier gives additional processing information about the bar code symbol generated. For example, it indicates whether a check-digit is generated for the bar code symbol. The meaning of the modifier values will vary depending on the type of bar code symbol. Refer to *Bar Code Object Content Architecture Reference* for a complete list. Valid values are:

AIM USS-39, Code 39 (3 of 9)	X'01' and X'02'
Modified Plessey	X'01' through X'09'
UPC/CGPC Version A	X'00'
UPC/CGPC Version B	X'00'
UPC — 2 digit supplemental	X'00'
UPC — 5 digit supplemental	X'00'
EAN-8 (includes JAN-short)	X'00'
EAN-13 (includes JAN-standard)	X'00'
Industrial 2-of-5	X'01' and X'02'
Matrix 2-of-5	X'01' and X'02'
Interleaved 2-of-5, AIM USS-I 2/5	X'01' and X'02'
AIM USS-Codabar, Codabar 2-of-7	X'01' and X'02'
Code 128, AIM USS-128	X'02'
EAN two-digit supplemental	X'00'
EAN five-digit supplemental	X'00'
POSTNET	X'00' through X'03'
Royal mail	X'00' and X'01'
Japan postal	X'00' and X'01'
Data matrix	X'00'
MaxiCode	X'00'
PDF417	X'00' and X'01'
Australia postal	X'00' through X'08'

**HRI Font**

Some bar code types have specific requirements for the type of HRI font used, and some bar code types do not allow HRI at all. Currently the only valid values for this parameter are:

DEFAULT_BCFONT	(for C)
DEFAULT-BCFONT	(for COBOL)
DEFFONT	(for RPG)

which means use the default for the device. All other values are ignored. The input encoding is controlled with the Set Input Codepage function (or the default).

| This parameter is ignored for Data Matrix, MaxiCode, and PDF417 type bar codes but a value must still be specified since all parameters are required on C function call.

**Color**

The color in which the bars of the symbol and the HRI data are presented. Valid values are:

BLUE
RED
MAGENTA or PINK
GREEN
CYAN or TURQ
YELLOW
BLACK
BROWN

## Begin Bar Code

MUSTARD  
DARKBLUE  
DARKGREEN  
DARKTURQ or DARKCYAN  
ORANGE  
PURPLE  
GRAY  
NONE  
MEDIUM\_COLOR  
DEFAULT\_COLOR

### Module width

Specifies the width in mils (thousandths of an inch) of the smallest defined bar code element. Some bar code symbologies refer to this value as the unit or X-dimension width. The widths of all symbol elements (bars and spaces) are normally expressed as multiples of the module width. Specify `DEFAULT_BCMODWIDTH` (for C), `DEFAULT-BCMODWIDTH` (for COBOL), or `DEFMWID` (for RPG) to use the default.

| This parameter is ignored for Data Matrix, MaxiCode, and PDF417 type bar codes but a value must  
| still be specified since all parameters are required on C function call.

### Element height

Specifies the height of the bar code symbol. The element height and height multiplier values are used to compute the total bar and space height of the bar code symbol. Valid values are:

0 to 32767  
-1

-1 indicates that the default element height of the presentation device is used.

| This parameter is ignored for Data Matrix, MaxiCode, and PDF417 type bar codes but a value must  
| still be specified since all parameters are required on C function call.

### Height multiplier

Specifies a value that, when multiplied by the element height, yields the total bar and space height presented. Valid values are 1 through 255.

| This parameter is ignored for Data Matrix, MaxiCode, and PDF417 type bar codes but a value must  
| still be specified since all parameters are required on C function call.

### Wide\_to\_narrow ratio

Specifies the ratio of the wide-element dimension to the narrow-element dimension for a two level bar code symbol. For example, if you want a ratio of 2.25 to 1, set this field to 225. You can specify a value of -1 to indicate that the device default should be used.

This parameter is ignored for POSTNET, EAN, UPC, Data Matrix, MaxiCode, and PDF417 type bar codes but a value must still be specified since all parameters are required on C function calls.

## Output Parameters

### Bar code handle

Handle used on subsequent Bar Code Data calls to place the actual bar code information on the page. The handle is also used by the End Bar Code call.

## Return Codes

- 1 The **outp\_bcHandle** value is null. You must specify a valid address for this parameter.
- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 10 Invalid bar code type specified. Pick a valid type from the list.
- 14 This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

This example shows an Interleaved 2-of-5 bar code, generates a check digit (modifier of X'02'), and uses the default font for the HRI (which is generated and placed above the bar code). The bar code is 12mm high, and placed into a presentation space 120mm x 50mm with a height multiplier of one and a default wide-to-narrow ratio.

```
rc = AFPBgnBarCode(PageHandle,"BCOCA 1",MM_2_U1440(120),MM_2_U1440(50),
    DEG_0,INTERLEAVED_2_OF_5_BC,0x02,DEFAULT_BCFONT,
    BLACK,DEFAULT_BCMODWIDTH,MM_2_U1440(12),
    1,-1,&BarCodeHandle);
```

```
rc = AFPBarCodeData(PageHandle,BarCodeHandle,"01234567890",
    MM_2_U1440(5),MM_2_U1440(5),
    TRUE,HRIOnTop,FALSE);
```

```
rc = AFPEndBarCode(PageHandle,&BarCodeHandle);
```

- **For the AFP Toolbox COBOL Interface:**

This example shows an Interleaved 2 of 5 barcode.

```
* Make the Interleaved 2 of 5 barcode
      MOVE "BCOCA 1"          TO AFP-BCOCA-NAME.
      MOVE 6803               TO AFP-AREA-XSIZE.
      MOVE 2834               TO AFP-AREA-YSIZE.
      MOVE DEG-0              TO AFP-OBJECT-ROTATION.
      MOVE INTERLEAVED-2-OF-5-BC TO AFP-BC-TYPE.
      MOVE 2                   TO AFP-BC-MODIFIER.
      MOVE DEFAULT-BCFONT     TO AFP-BC-FONT-ID.
      MOVE BLACK              TO AFP-BC-COLOR.
      MOVE DEFAULT-BCMODWIDTH TO AFP-BC-MODWIDTH.
      MOVE 680                TO AFP-BC-ELHEIGHT.
      MOVE 1                   TO AFP-BC-HEIGHTMULT.
      MOVE -1                  TO AFP-BC-WIDETONARROW.
      PERFORM AFPBBCD.

* Add the barcode data
      MOVE "01234567890" TO AFP-BARCODE-DATA.
      MOVE 283           TO AFP-BCSYM-XPOS.
      MOVE 283           TO AFP-BCSYM-YPOS.
      MOVE TRU           TO AFP-BCHRI-PRESENT.
      MOVE HRI-ON-TOP    TO AFP-BCHRI-POS.
      MOVE FALS          TO AFP-BC-ASTERISK.
      PERFORM AFPBCDAT.
      PERFORM AFPEBCD.
```

- **For the AFP Toolbox RPG Interface:**

```
*Provide general information about the bar code generated.
C          MOVEL      'BCOCA1'      AFPBCNAM
C          CAT(P)     X'00':0       AFPBCNAM
C          Z-ADD      7200          AFPBCXSIZ
C          Z-ADD      2880          AFPBCYSIZ
C          Z-ADD      DEG0          AFPBCRTAT
C          Z-ADD      INTRL20F5     AFPBCTYPE
C          MOVE       x'02'         AFPBCMOD
C          Z-ADD      1              AFPBCFONT
C          Z-ADD      BLACK         AFPBCCOL
C          Z-ADD      -1            AFPBCMODW
C          Z-ADD      -1            AFPBCELHT
C          Z-ADD      1             AFPBCHMUL
C          Z-ADD      -1            AFPBCW2N
C          EXSR       AFPBBCCR

*Add data to above bar code and specify how the HRI should be presented.
C          MOVEL      '12345678'    AFPBCDATA
C          CAT(P)     X'00':0       AFPBCDATA
C          Z-ADD      1              AFPBCXPOS
C          Z-ADD      1              AFPBCYPOS
```

## Begin Bar Code

C	MOVE	TRU	AFPBCHRIP
C	MOVE	HRIONBOT	AFPBCHRPS
C	Z-ADD	FALS	AFPBCASTR



## Begin Box

### Function

Begins a box at the current location. The box is not actually drawn until the corresponding End Box is issued. The box must be ended before you end the page. Ensure that the box you have specified fits on the logical page.

**Note:** To provide device-independent formatting, the output data stream produced by AFP Toolbox has a resolution of 1440 ppi. The printer rounds positioning information from 1440ths of an inch to the nearest pel (240, 300, and so forth). This might cause one pel rounding errors that could be evident in rule widths and rule intersections. To avoid this problem, use the GOCA box functions (see “GOCA Draw Box” on page 272).

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPBgnBox(
    TBHANDLE    in_pageHandle,
    ushort      in_rulethickness,
    ushort      in_boxwidth,
    TBHANDLE*    outp_boxHandle
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPBBBOX.
    CALL "CBLBgnBox"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-RULE-THICKNESS
                AFP-BOX-WIDTH
            BY REFERENCE
                AFP-BOX-HANDLE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLBgnBox" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

#### For the AFP Toolbox RPG Interface:

```
DAFPBBBOX      PR          10I 0 EXTPROC('AFPBgnBox')
D AFPPAGHDL          *  VALUE
D AFPRULTHK          5I 0  VALUE
D AFPBOXWID          5I 0  VALUE
D AFPBOXHDL          *
```

## Input Parameters

### Page handle

The handle for the page. Page handle is returned by the Begin Page call.

### Rule thickness

The width of the rule used for the sides of the box.

### Box width

Width of the box. The width of the box is calculated from the left side of the left most rule to the left side of the right most rule. The box width includes the width of the left side rule but not the right side

## Begin Box

rule. Zero is the default. If the width is a positive number, the box is drawn to the right of the current position; otherwise it is drawn to the left of the current position.

## Output Parameters

### Box handle

The handle for this box. Box handle is used on the End Box call when you are finished drawing this box. The box depth is calculated at that time.

## Return Codes

- 1 The **outp\_boxHandle** value is null. You must specify a valid address for this parameter.
- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

### • For the AFP Toolbox C Library:

Begin a box at the current location with 135mm wide and .5mm thick sides.

```
TBHANDLE currentbox;
short boxrule, boxwidth;
boxrule = MM_2_U1440 (.5);
boxwidth = MM_2_U1440 (135);
AFPBgnBox(currentpage, boxrule, boxwidth, &currentbox);
rc = AFPVMove(currentpage, IN_2_U1440(4.5));
rc = AFPEndBox(currentpage, &currentbox);
```

### • For the AFP Toolbox COBOL Interface:

This example positions the box at one inch across, one inch down, and draws a box 5 inches wide by 4.5 inches deep.

```
* Set the box position to 1 inch by 1 inch
MOVE 1440 TO AFP-X-COORDINATE.
MOVE 1440 TO AFP-Y-COORDINATE.
MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
PERFORM AFPSPOS.
* Start a box that will be 5 inches wide by 4.5 inches deep
MOVE 50 TO AFP-RULE-THICKNESS.
MOVE 7200 TO AFP-BOX-WIDTH.
PERFORM AFPBBOX.
* Move the position down vertically 4.5 inches
MOVE 6480 TO AFP-Y-COORDINATE.
PERFORM AFPVMOVE.
* End the box
PERFORM AFPEBOX.
```

### • For the AFP Toolbox RPG Interface:

\*Begins a box.

C	Z-ADD	20	AFPRULTHK
C	Z-ADD	4320	AFPBOXWID
C	EXSR	AFPBBOXR	

## Begin Color

### Function

Begins a colored block at the current location. The block is not actually placed until the corresponding End Color is issued. This function requires Print Services Facility (PSF) and hardware microcode support.

**Note:** Use of this function requires microcode support in your printer.

### Syntax

Create a blue rectangle two inches wide using hilight color space with 25% black coverage:

#### For the AFP Toolbox C Library:

```
int AFPBgnColor(
    TBHANDLE    in_pageHandle
    ColorSpace  in_colorType
    MODCAColors in_color
    ulong       in_colspec1
    ulong       in_colspec2
    ulong       in_colspec3
    ulong       in_colspec4
    ulong       in_colorwidth
    TBHANDLE    outp_colorHandle
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPBCLR.
  CALL "CBLBgnColor"
  USING
  BY VALUE
    AFP-PAGE-HANDLE,
    AFP-COLOR-TYPE,
    AFP-COLOR,
    AFP-COLSPEC1,
    AFP-COLSPEC2,
    AFP-COLSPEC3,
    AFP-COLSPEC4,
    AFP-COLOR-WIDTH
  BY REFERENCE
    AFP-COLOR-HANDLE,
  RETURNING
    AFP-RET-CODE.
  MOVE "CBLBeginColor" TO AFP-ERRDATA.
  PERFORM CHKSUCC.
```

## Input Parameters

### Page handle

The handle for the page. Page handle is returned by the Begin Color call.

### Color type

The type of color space to generate. Valid values are:

#### RGB\_COLOR (for C) or RGB-COLOR (for COBOL)

The color value is specified as three intensity levels for the red, green, and blue components of the color. These numbers are given for color specification values 1, 2, and 3. The fourth specification is ignored and should be specified as 0.

## Begin Color

### CMYK\_COLOR (for C) or CMYK-COLOR (for COBOL)

The color value is specified as four intensity levels for the cyan, magenta, yellow, and black components of the color. These are color specifications 1-4.

### HILIGHT\_COLOR (for C) or HILIGHT-COLOR (for COBOL)

Defines a request for presentation device to generate a highlight color. Specify a MO:DCA Color value for the highlight color. Color specification 1 is the percentage coverage for the specified color, and color specification 2 is the percentage of black that is added to the coverage (can be 0). Color specifications 3 and 4 should be 0.

### CIELAB\_COLOR (for C) or CIELAB-COLOR (for COBOL)

Specified with three components. Color specification 1 is the luminance, and color specifications 2 and 3 are the chrominance differences. Color specification 4 should be 0.

### OCA\_COLOR (for C) or OCA-COLOR (for COBOL)

Defines a request for presentation device to generate a highlight color. Specify a MO:DCA Color value for the highlight color. The presentation device will produce printed colors using a color mapping table (see PSF documentation for your environment). Color specifications 1-4 should be 0.

## Color

Any of the defined MO:DCA color values: BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. This value is only used for Highlight or OCA color spaces but always must be specified due to C language conventions.

### Color specification 1-4

Usage depends on the color space type, see descriptions above.

### Color width

The width of the color area.

## Output Parameters

### Color handle

The handle for this colored area. Color handle is used on the End Color call when you are finished coloring this area. The width of the colored area is calculated at that time.

## Return Codes

- 1 The **Color handle** value is null. You must specify a valid address for this parameter.
- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

Create a blue rectangle two inches wide using hilight color space with 25% black coverage:

- **For the AFP Toolbox C Library:**

```
rc = AFPBgnColor(currentpage,HILIGHT_CS,BLUE,
                 0,25,0,0,2880,&currentcolor);
rc = AFPVMove(currentpage,2880);
rc = AFPEndColor(currentpage,&currentcolor);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE "cobl goca" TO AFP-DOC-NAME.
MOVE HILIGHT-CS TO AFP-COLOR-TYPE.
MOVE BLUE TO AFP-COLOR.
MOVE 0 TO AFP-COLSPEC1.
MOVE 25 TO AFP-COLSPEC2.
MOVE 0 TO AFP-COLSPEC3.
MOVE 0 TO AFP-COLSPEC4.
```

```
    MOVE 2880 TO AFP-COLOR-WIDTH.  
PERFORM AFPBCLR.  
    MOVE 2880 TO AFP-Y-COORDINATE.  
PERFORM AFPVMOVE.  
PERFORM AFPECLR.
```

---

# Begin Document

## Function

Begins a document and specifies the name of the document used in the output data stream as well as the type of output produced.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPBgnDoc(
    TBHANDLE    in_sessHandle,
    char*        in_docName,
    char*        in_comment,
    OUTTYPE     in_outputtype,
    char*        in_outputfilename,
    byte*        in_outputbuffer,
    TBHANDLE*    outp_docHandle
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPBDOC.
    CALL "CBLBgnDoc"
        USING
            BY VALUE
                AFPAPI-HANDLE
            BY CONTENT
                AFP-DOC-NAME
                AFP-DOC-COMMENT
            BY VALUE
                AFP-OUTPUT-TYPE
            BY CONTENT
                AFP-OUTPUT-FILENAME
                AFP-BUFFER
            BY REFERENCE
                AFP-DOCUMENT-HANDLE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLBgnDoc" TO AFP-ERRDATA.
    PERFORM CHKSUC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPBDOC          PR          10I 0 EXTPROC('AFPBgnDoc')
D  AFPAPIHDL              *    VALUE
D  AFPDOCNAM              250
D  AFPDOCCMT              250
D  AFPOUTTYP              10I 0 VALUE
D  AFPOUTFLE              150
D  AFPBUF                 8205
D  AFPDOCHDL              *
```

## Input Parameters

### Session handle

The handle for the session. Session handle is returned by the Begin Session call.

### Name of document

The name can be up to 250 bytes long. If the name is longer than 250 bytes, it is truncated.

**Comment**

Up to 250 bytes of data to place as a comment in the output. If the comment is longer than 250 bytes, it is truncated.

**Type of output**

Output to a buffer or output to a file. Valid values are:

- `TO_OUTPUT_FILE` (C) or `FILED` (COBOL, RPG) to write the output document to the file specified with the output filename parameter.
- `TO_OUTPUT_BUFFER` (C) or `BUFFERED` (COBOL, RPG) to have the AFP structured fields returned to the application program for processing.
- `TO_BIT_BUCKET` (C) or `BITBUCKET` (COBOL, RPG) to have the page discarded instead of being converted into AFP. This function is used for counting pages without actually producing any output.

**Name of output file**

Name of the file where the AFP output is written. The output file name is the same as the actual file name except in the MVS environment where it is the DDNAME. This value is ignored if buffered is specified for output type.

**Note:** If you are writing to an HFS file on an OS/390 system, you must still either write to a DD name or specify buffered output and write to the HFS file directly from your application. If using buffered output, you must allocate the HFS file as a binary file, for example:

```
static char myoutput[] = "/u/jls/buffer.afp";
out = fopen(myoutput,"wb");
```

**Output buffer**

The address of the buffer where the AFP output is written. This value is ignored if a file is specified for output type. The output buffer must be a minimum of 8205 bytes or the size of the largest structured field for any inline resources.

**Output Parameters****Document handle**

The handle for the document. The document handle is used on subsequent calls that place information at the document level, for example, Begin Group, Begin Page, and so forth. It is also used on the corresponding End Document call.

**Return Codes**

- 1 The **outp\_docHandle** value is null. You must specify a valid address for this parameter.
- 2 Session handle is null or invalid. Make sure you have issued Begin Session before you call Begin Document.
- 19 An error occurred while trying to open the output file specified with **in\_outputfilename**.

**Sample Function Call**

- **For the AFP Toolbox C Library:**

This example shows how to produce either buffered or filed output. In MVS, the output file is a DD name. In other environments, it is the actual file name. For filed output, the AFP Toolbox opens the file for you. If you are using buffered output, you must open the file for binary output as shown below.

**Note:** If you are running on MVS, you need to #define the MVS symbol in your application or at the time the file is compiled.

```
#ifndef MVS
static char outfile[] = "e:\\tbxout\\filetst.afp";
static char myoutput[] = "e:\\tbxout\\bufftst.afp";
#else
static char outfile[] = "BUFTSTDD";
```

## Begin Document

```
static char myoutput[] = "bufftst.afp";
#ifdef
byte mybuff[8*1024]; /* buffer to contain AFP records */ FILE* out;
/* if you want buffered output, change the next line to TO_OUTPUT_BUFFER*/
OUTTYPE output = TO_OUTPUT_FILE;
if (output == TO_OUTPUT_BUFFER)
{
#ifdef MVS
out = fopen(myoutput, "wb");
#else
/* file should be allocated with lrecl 8205 */
out = fopen(myoutput, "wb, recfm=*, type=record");
#endif
if (errno != 0)
{
perror("Error occurred while opening file.\n");
exit (1);
}
} // end buffered output
rc = AFPBgnDoc (SessionHandle,"SAMPLE","MY TEST SAMPLE",
               output,outfile,mybuff,&DocumentHandle) ;
rc = AFPBgnPage (DocumentHandle,&pagecount,&PageHandle) ;
...
rc = AFPEndPage (&PageHandle) ;

/* after each page is ended the AFP output must be retrieved */
/* the 'morestuff' flag will be set to false by Toolbox when */
/* there are no more output records to process. */
if (output == TO_OUTPUT_BUFFER)
{
morestuff = TRUE;
do
{
rc = AFPGetBuffer(&PageHandle,&bufflen,&morestuff);
if (morestuff)
count = fwrite(mybuff,1,bufflen,out); /* write to output file */
} // end do while
while(morestuff);
} // end if buffered output
rc = AFPEndDoc (&DocumentHandle) ;
if (output == TO_OUTPUT_BUFFER) /* get EDT and any other SFs */
{
morestuff = TRUE;
do
{
rc = AFPGetBuffer(&DocumentHandle,&bufflen,&morestuff);
if (morestuff)
count = fwrite(mybuff,1,bufflen,out); /* write to output file */
} // end do while
while(morestuff);
} // end if buffered output
```

- **For the AFP Toolbox COBOL Interface:**

Begins a document whose output in MVS goes to DD name MYOUTPT.

```
MOVE "COBOL TEST DOC" TO AFP-DOC-NAME.
MOVE "And a comment" TO AFP-DOC-COMMENT.
MOVE FILED TO AFP-OUTPUT-TYPE.
MOVE "MYOUTPT"
    TO AFP-OUTPUT-FILENAME.
PERFORM AFPBDOC.
```

For a COBOL example of buffered output, see "CBLGetBuffer Program" on page 382.

- **For the AFP Toolbox RPG Interface:**

Begins a document whose output goes to an OS/400 file.



```
* Set the output characteristics
C      DTA(3)      CAT(P)  x'00':0  AFPOCNAM
C      DTA(4)      CAT(P)  x'00':0  AFPOCCMT
C      Z-ADD       FILED    AFPOUTTYP
C      DTA(5)      CAT(P)  x'00':0  AFPOUTFLE
C      EXSR        AFPBDOCR

...
**CTDATA DTA
Page 1                                1
HELLO WORLD                          2
RPG TEST DOC                         3
And a comment                        4
/QSYS.LIB/MYLIB.LIB/AFPOUT.FILE/RPGTEST.MBR 5
```

---

# Begin Field

## Function

Begins a new field of data in a table.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPBeginField(
    TBHANDLE    in_tblHandle,
    uint        in_fieldid
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPBFLD.
    CALL "CBLBeginField"
    USING
    BY VALUE
    AFP-TABLE-HANDLE
    AFP-FIELD-ID
    RETURNING
    AFP-RET-CODE.
    MOVE "CBLBeginField" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

## Input Parameters

### Table Handle

The handle for the associated table, returned from the Begin Table call.

### Field ID

The ID of the row definition returned from the associated Define Field call.

## Return Codes

- 1** The table object handle is invalid or null. Call Begin Table before issuing this call.
- 25** The field ID is invalid. Call Define Field before issuing this call.

## Sample Function Call

This command begins a field, using the ID returned from Define Field for formatting.

- **For the AFP Toolbox C Library:**

```
/* Begin Header Field 1 */
RetCode = AFPBeginField(hTbl, Hdrfld1);
checkForError(RetCode,"Begin Field Hdrfld1\n");

/* Put Text in Header Field 1 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Date");

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld1,szTextstring, headingsFont);
checkForError(RetCode,"PutText Field Hdrfld1\n");

/* End Header Field 1 */
RetCode = AFPEndField(hTbl);
checkForError(RetCode,"End Field Hdrfld1\n");
```

- **For the AFP Toolbox COBOL interface:**

```
MOVE FIELDH1 TO AFP-FIELD-ID.  
PERFORM AFPBFLD.  
MOVE TIM14BOLD TO AFP-FONT-ID.  
PERFORM AFPSFONT.
```

```
MOVE NULLS TO AFP-CHARACTER-STRING.  
MOVE "Test" TO AFP-CHARACTER-STRING.  
MOVE 4 TO AFP-STRING-LENGTH.  
PERFORM AFPPCHS.
```

```
PERFORM AFPEFLD.
```

# Begin Group

## Function

Begins a logical group of pages. Page groups can be used for indexing and retrieving information in the document by online viewing products such as AFP Workbench and OnDemand.

Pages separated with Begin and End Group can be indexed with the Attribute Name and Attribute Value parameters of the Put Tag procedure call. The group name should be unique within a document.

**Note:** Groups of pages cannot be nested or overlapped, each group must be ended before another can begin.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPBgnGroup(
    TBHANDLE    in_docHandle,
    char*        in_groupName
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPBGRP.
    CALL "CBLBgnGroup"
        USING
            BY VALUE
                AFP-DOCUMENT-HANDLE
            BY CONTENT
                AFP-GROUP-NAME
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLBgnGrp" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPBGRP          PR          10I 0  EXTPROC('AFPBgnGroup')
D  AFPDOCHDL          *  VALUE
D  AFPGRPNAM          250
```

## Input Parameters

### Document handle

The handle for the document. Document handle is returned by the Begin Document call.

### Group name

The name of the group to start. The group name should be unique within a document. The maximum number of characters in the group name is 250. If the name is longer than 250, it is truncated. Blanks are allowed as part of the group name.

## Return Codes

- 3** The document handle is missing or is not valid. Make sure you have called Begin Document before issuing this call.
- 11** Nested groups or overlapped groups are not allowed. End the previous group before beginning a new group.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Identifying the start of a group of pages associated with a particular account number.

```
strcpy(GroupName, "Account Number");
rc = AFPBgnGroup(DocHandle, GroupName);
```

- **For the AFP Toolbox COBOL Interface:**

Specifies the beginning of a group of pages.

```

      MOVE "PUTTAG" TO AFP-DOC-NAME.
      MOVE "NO COMMENT" TO AFP-DOC-COMMENT.
      MOVE FILED TO AFP-OUTPUT-TYPE.
      MOVE "MYOUTPT"
        TO AFP-OUTPUT-FILENAME.
      PERFORM AFPBDOC.
* Begin the group; put 1 page in this group
      MOVE "Group1" TO AFP-GROUP-NAME.
      PERFORM AFPBGRP.
* Put in the tag
      MOVE "GroupTagName" TO AFP-TAG-NAME.
      MOVE "GroupTagValue" TO AFP-TAG-VALUE.
      SET AFP-CURRENT-HANDLE TO AFP-DOCUMENT-HANDLE.
      PERFORM AFPPTAG.
* Begin the page
      MOVE "PAGE1" TO AFP-PAGE-NAME.
      PERFORM AFPBPAGE.
* Now put the tag at the page level
      MOVE "Policy Number" TO AFP-TAG-NAME.
      MOVE "4327894-8324" TO AFP-TAG-VALUE.
      SET AFP-CURRENT-HANDLE TO AFP-PAGE-HANDLE.
      PERFORM AFPPTAG.
* Put some text on the page
* End the page
      PERFORM AFPEPAGE.
* End the group
      PERFORM AFPEGRP.
* Other pages could go here
* End the document
      PERFORM AFPEDOC.
```

- **For the AFP Toolbox RPG Interface:**

Specifies the beginning of a group of pages.

```
*Begins a logical group of pages.
C          MOVEL      'Group 1'      AFPGRPNAM
C          EXSR       AFPBGRPR
C          MOVEL      Tag1           AFPTAGNAM
C          MOVEL      'Tag Value'    AFPTAGVAL
C          EXSR       AFPPTAGR
```

---

# Begin Image

## Function

Begin building an image object for inclusion in the page at the current position. Data is then placed in the image object using Image Data calls. The image data can be an IOCA or IM1 image. Page segments containing IM1 image data can only use the POSITION\_MOPT mapping option. Do not issue this call within a paragraph or a table.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPBgnImage(
    TBHANDLE      in_pageHandle,
    char*          in_IOCAName,
    uint           in_hr,
    uint           in_vr,
    uint           in_hs,
    uint           in_vs,
    ulong          in_areaxsize,
    ulong          in_areaysize,
    ulong          in_objareaxpos,
    ulong          in_objareaypos,
    MappingOption  in_objmapopt,
    DegRot         in_objrotation,
    TBHANDLE*      outp_imageHandle
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPBIMG.
    CALL "CBLBgnImage"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
            BY CONTENT
                AFP-IOCA-NAME
            BY VALUE
                AFP-HRES
                AFP-VRES
                AFP-HSIZE
                AFP-VSIZE
                AFP-AREA-XSIZE
                AFP-AREA-YSIZE
                AFP-AREA-XPOS
                AFP-AREA-YPOS
                AFP-OBJECT-MAPPING-OPTION
                AFP-OBJECT-ROTATION
            BY REFERENCE
                AFP-IMAGE-HANDLE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLBgnImage" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```

DAFPBIMAGE      PR          10I 0 EXTPROC('AFPBgnImage')
D AFPPAGHDL      *          VALUE
D AFPIONAM       250
D AFPIHRES       10I 0 VALUE
D AFPIVRES       10I 0 VALUE
D AFPIHSIZE      10I 0 VALUE
D AFPIVSIZE      10I 0 VALUE
D AFPIAXSIZ      10I 0 VALUE
D AFPIAYSIZ      10I 0 VALUE
D AFPIAXPOS      10I 0 VALUE
D AFPIAYPOS      10I 0 VALUE
D AFPIAMAP       10I 0 VALUE
D AFPIARTAT      10I 0 VALUE
D AFPIMGHDL      *

```

## Input Parameters

### Page handle

The handle for the page. Page handle is returned by the Begin Page call.

### IOCA name

The name of the object as it is used in the output data stream. The maximum number of characters in the name is 250. If the name is longer than 250, it is truncated.

### Horizontal and vertical resolution

These parameters specify the resolution (pels per ten inches) of the image data. You can specify resolutions between 1 and 32767, but your printer must be able to support what you specify. You must specify the actual values for which the image was created. For example, if you scanned an image at 200 PPI, specify 2000 for these parameters.

### Horizontal and vertical size

These parameters specify the size of the image object as it should appear when it is printed. The values must be given in units of the image specified with the horizontal and vertical resolution parameters.

### Area size

These parameters specify the size of the area for the image data.

The areaxsize and areaysize are the width and depth of the image on the printed page respectively in 1440ths of an inch.

### Area position

These coordinates specify the offset within the object area of the object content in 1440ths of an inch. They are only used with a mapping option if one of these is used:

POSITION\_MOPT or POSITION\_AND\_TRIM\_MOPT (for C)  
 POSITION-DEFAULT or POSITION-AND-TRIM (for COBOL)  
 POSDEFLT or POSTRIM (for RPG)

See Figure 25 on page 336 for an example of positioning an object.

### Mapping option

This parameter specifies the mapping for the content of the area. Valid values are:

#### **POSITION\_MOPT (for C), POSITION-DEFAULT (for COBOL), or POSDEFLT (for RPG)**

Position the upper left corner of the object's presentation space with the object's content origin.

#### **POSITION\_AND\_TRIM\_MOPT (for C), POSITION-AND-TRIM (for COBOL), or POSTRIM (for RPG)**

Position the object at the location specified with ObjectAreaPosition within the dimensions specified with ObjectAreaSize and trim what falls outside the area.

## Begin Image

### **SCALE\_TO\_FIT\_MOPT (for C), SCALE-TO-FIT (for COBOL), or SCALE2FIT (for RPG)**

Center the object within the area dimension specified with ObjectAreaSize and scale the object to fit within the area.

### **CENTER\_AND\_TRIM\_MOPT (for C), CENTER-AND-TRIM (for COBOL), or CENTRTRIM (for RPG)**

Center the object within the area dimension specified with ObjectAreaSize and trim what falls outside the area.

### **MIGRATION\_MAPPING\_41\_MOPT (for C), MIGRATION-MAP-1 (for COBOL), or POINT2PEL (for RPG)**

This mapping is for IM image migration. The origin of the IO image presentation space is positioned at the origin of the object area. Each image point in the presentation space is mapped to a presentation device pel. Any portion of the image that falls outside the object area is trimmed.

### **MIGRATION\_MAPPING\_42\_MOPT (for C), MIGRATION-MAP-2 (for COBOL), or DOUBLEDOT (for RPG)**

This mapping is for IM image migration. The origin of the IO image presentation space is positioned at the origin of the object area. Each image point in the presentation space is doubled in both directions, resulting in four new image points. The four new image points are then mapped to presentation device pels. Any portion of the image that falls outside the object area is trimmed.

### **REPLICATE\_AND\_TRIM\_MOPT (for C), REPLCT-AND-TRIM (for COBOL), or REPLCTRIM (for RPG)**

Position the data object presentation space is in the object area so that its origin is coincident with the origin of the object area and its size is unchanged. The data object presentation space or graphics presentation space window in the case of GOCA is then replicated in the X and Y directions of the object area until the object area is filled.

## Object rotation

Specifies the clockwise rotations of the object area from the x axis rotation of the page. Valid values are:

### **DEG\_0 (for C), DEG-0 (for COBOL), or DEG0 (for RPG)**

The object area is not rotated.

### **DEG\_90 (for C), DEG-90 (for COBOL), or DEG90 (for RPG)**

The object area is rotated 90 degrees clockwise.

### **DEG\_180 (for C), DEG-180 (for COBOL), or DEG180 (for RPG)**

The object area is rotated 180 degrees clockwise.

### **DEG\_270 (for C), DEG-270 (for COBOL), or DEG270 (for RPG)**

The object area is rotated 270 degrees clockwise.

## Output Parameters

### Image handle

The handle for the image. Image handle is used on the Image Data and End Image calls.

## Return Codes

- 1 The **outp\_imageHandle** value is null. You must specify a valid address for this parameter.
- 4 Page handle is missing or is not valid. Make sure you have called Begin Page before issuing this call.
- 14 This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.

## Sample Function Call

### • For the AFP Toolbox C Library:

This example begins an image object with a resolution of 240 pels (horizontal resolution of 2400 pels per 10 inches) by 120 pels (vertical resolution of 1200 pels per 10 inches). Its size is 8.5 inches (2040



units at 240 ppi) by 11 inches (1320 units at 120 ppi). The object area size is 3.67 inches square (5280 by 5280, given in units of 1440 ppi regardless of the image resolution). The object area position is (0,0) relative to the start of the image, which will be placed at the current position on the page (in this example, that is at (0,0) since we have not issued an AFPSetPosition call). The rotation of the object relative to the page is 0.

```
char inputFileName1[] = argv[];    // input image file name
byte iocadata[3000];

int process(char* file, byte* iocout, int* len);

process(inputFileName1,iocadata,&len);    // read the image file

rc = AFPBgnImage(currentpage,"IOCA1",2400,1200,2040,1320,5280,5280,0,0,
SCALE_TO_FIT_MOPT,DEG_0,&currentimage);
rc = AFPIImageData(currentpage,currentimage,iocadata,len,MMR,1,0);
rc = AFPEndImage(currentpage,&currentimage);

//returns IPD data from pseg
int process(char* file, byte* iocout, int* len)
{
char inbuf[4096];                //buffer to hold pseg
int sflength = 0, datalength = 0;
int j = 0,l = 0,k;

FILE *fp = fopen(file,"rb");

fread(inbuf,4096,1,fp);

    {while (inbuf[j] == '\x5a' )

        //determine length of structured field
        { sflength = inbuf[j+1] * 256 + inbuf[j+2];

if (inbuf[j+3] == '\xd3' &&                //look for IPD data
inbuf[j+4] == '\xee' &&
inbuf[j+5] == '\xfb' &&
inbuf[j+9] == '\xfe' &&
inbuf[j+10] == '\x92')

    //determine length of data
    {datalength = inbuf[j+11] * 256 + inbuf[j+12];

    for (k = 0; k < datalength; k+=1)                //save data bytes
        {iocout[l] = inbuf[j+13+k];
        l+=1;}
    }

    j+=sflength+1;                //go to next sf
    }

}

*len = l;
fclose(fp);
return 0;
}
```

- **For the AFP Toolbox COBOL Interface:**

This example produces an IOCA image object with a resolution of 300 pels horizontally and vertically. The object size is 600 pels (2 inches) by 450 (1.5 inches). The object area size is four inches by three inches. The mapping option is SCALE-TO-FIT. The object area position is (0,0) and the rotation is 90 degrees.

**Note:** The image data in this example ("abcdefghij") is an example only and does not produce a valid image. You would use your own image data to produce your own image.

## Begin Image

```

      MOVE "IOCA1"          TO AFP-IOCA-NAME.
      MOVE 3000             TO AFP-HRES.
      MOVE 3000             TO AFP-VRES.
      MOVE 600              TO AFP-HSIZE.
      MOVE 450              TO AFP-VSIZE.
      MOVE 1200             TO AFP-AREA-XSIZE.
      MOVE 900              TO AFP-AREA-YSIZE.
      MOVE 0                TO AFP-AREA-XPOS.
      MOVE 0                TO AFP-AREA-YPOS.
      MOVE SCALE-TO-FIT     TO AFP-OBJECT-MAPPING-OPTION.
      MOVE DEG-90          TO AFP-OBJECT-ROTATION.
PERFORM AFPBIMG.
      MOVE "abcdefghij"     TO AFP-IMAGE-BUFFER.
      MOVE 26               TO AFP-BUFFER-LENGTH.
      MOVE MMR              TO AFP-COMPRESSION-TYPE.
      MOVE 1                TO AFP-RECORDING-ALG.
      MOVE 0                TO AFP-BIT-ORDER.
PERFORM AFPIDATA.
PERFORM AFPEIMG.

```

- **For the AFP Toolbox RPG Interface:**

This example produces an IOCA image object with a resolution of 300 pels horizontally and vertically. The object size is 600 pels (2 inches) by 450 (1.5 inches). The object area size is four inches by three inches. The mapping option is SCALE2FIT. The object area position is (0,0) and the rotation is 90 degrees.

**Note:** The image data in this example ("abcdefghij") is an example only and does not produce a valid image. You would use your own image data to produce your own image.

```

C          Z-ADD      3000      AFPIHRES
C          Z-ADD      3000      AFPIVRES
C          Z-ADD      6000      AFPIHSIZE
C          Z-ADD      4500      AFPIVSIZE
C          Z-ADD      5760      AFPIAXSIZ
C          Z-ADD      4320      AFPIAYSIZ
C          Z-ADD      SCALE2FIT AFPIAMAP
C          Z-ADD      DEG90     AFPIARTAT
C          Z-ADD      AFPBIMR
*Include image data records into the above image.
C          MOVE-L     'abcde'   TEXT10
C          MOVE       'fghij'   TEXT10
C          TEXT10      CAT(P)    X'00':0      AFPIMBUFF
C          CAT(P)      X'00':0      AFPIMBUFF
C          Z-ADD      10         AFPIMBLEN
C          Z-ADD      RAW        AFPIMCOMP
C          Z-ADD      1         AFPIMRECD
C          EXSR       AFPIDATR
*Ends the image
C          EXSR       AFPEIMGR

```

## Begin Page

### Function

Begins a logical page and names it. The initial current position is at the page origin (the top left corner of the logical page).

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPBgnPage(
    TBHANDLE    in_docHandle,
    char*       in_pageName,
    TBHANDLE*    outp_pageHandle
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFBPAGE.
    CALL "CBLBgnPage"
        USING
            BY VALUE
                AFP-DOCUMENT-HANDLE
            BY CONTENT
                AFP-PAGE-NAME
            BY REFERENCE
                AFP-PAGE-HANDLE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLBgnPage" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPBPAGE      PR          10I 0 EXTPROC('AFPBgnPage')
D  AFPDOCHDL          *    VALUE
D  AFPPAGNAM          250
D  AFPPAGHDL          *
```

## Input Parameters

### Document handle

The handle for this document. Document handle is returned by the Begin Document call.

### Name of page

The name can be up to 250 bytes long. If the name is longer than 250, it is truncated. The page name is used on the following structured fields: Begin Page, Begin Active Environment, Begin Presentation Text, End Page, End Active Environment, and End Presentation Text.

## Output Parameters

### Page handle

The handle for the page. Page handle is used on subsequent calls that place information at the page level, for example, Begin Image, Begin Paragraph, and so forth. It is also used on the corresponding End Page call.

## Return Codes

- 1 The **outp\_pageHandle** value is null. You must specify a valid address for this parameter.

## Begin Page

- 3      The document handle is missing or is not valid. Make sure you have called Begin Document before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Begin a page in the current document.

```
strcpy(PageName, "My Next Page");  
rc = AFPBgnPage(DocHandle, PageName, &PageHandle);
```

- **For the AFP Toolbox COBOL Interface:**

This example starts a logical page with the name of 'PAGE 1'.

```
MOVE "PAGE1" TO AFP-PAGE-NAME.  
PERFORM AFPBPAGE.
```

- **For the AFP Toolbox RPG Interface:**

This example starts a logical page with the name of 'PAGE 1'.

```
C      DTA(1)          CAT(P)    x'00':0      AFPPAGNAM  
* The above line places the DTA(1) data in AFPPAGNAM and adds a Hex'00' as  
C      EXSR           AFPBPAGER
```

```
...  
**CTDATA DTA  
Page 1
```

1

## Begin Paragraph

### Function

Begins a paragraph and specifies the formatting information used when placing text into the paragraph with Put Text calls.

Calls that change the current position or font, except for the Put Text call, are not allowed within a paragraph.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPBgnPgraph(
    TBHANDLE          in_pageHandle,
    ushort            in_paragraph_width,
    AlignmentOption    in_text_alignment,
    ushort            in_linespacing,
    ushort            in_max_pdepth
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPPBPARA.
    CALL "CBLBgnPgraph"
        USING
        BY VALUE
        AFP-PAGE-HANDLE
        AFP-PARA-WIDTH
        AFP-FORMAT-OPTION
        AFP-LINE-SPACING
        AFP-PARA-MAXDEPTH
    RETURNING
        AFP-RET-CODE.
    MOVE "CBLBgnPgraph" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPBPARA      PR          10I 0 EXTPROC('AFPBgnPgraph')
D AFPPAGHDL          *  VALUE
D AFPPARWTH          10I 0  VALUE
D AFPFMTOPT          10I 0  VALUE
D AFPLINSPC          10I 0  VALUE
D AFPPARMAX          10I 0  VALUE
```

## Input Parameters

### Page handle

The handle for this page. Page handle is returned by the Begin Page call.

### Paragraph width

The width of the paragraph.

### Text alignment

Specifies how the text in the paragraph should be formatted. The valid values are:

**LEFT\_ALIGN (for C), LEFT-ALIGN (for COBOL), or LEFTALGN (for RPG)**

Text is left aligned.

**RIGHT\_ALIGN (for C), RIGHT-ALIGN (for COBOL), or RIGHTALGN (for RPG)**

Text is right aligned.

## Begin Paragraph

**CENTER\_ALIGN (for C), CENTER-ALIGN (for COBOL), or CENTRALGN (for RPG)**

Text is centered.

**JUSTIFY\_ALIGN (for C), JUSTIFY-ALIGN (for COBOL), or JUSTFYALN (for RPG)**

Text is justified.

## Line spacing

Amount of vertical space to move from one line of text in the paragraph to the next. Vertical space is measured from the baseline of text to the next baseline. If you want to use the default line spacing value from the largest font in the line, specify:

**DEFAULT\_LSP (for C), DEFAULT-LPS (for COBOL), or DEFLSP (for RPG)**

for the line spacing parameter.

## Maximum paragraph depth

Maximum amount of vertical space to use when placing text into this paragraph. When the maximum depth is exceeded, a return code is issued on the Put Text call whose text does not fit in the paragraph. The depth of the paragraph is calculated from the baseline of the first line of text to the baseline of the last line of text that fits.

## Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 16 Nested paragraphs are not allowed. Issue End Paragraph before you begin another paragraph.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Create a paragraph 125mm wide by up to 25mm deep with justified text and place text in it:

```
char * remainingstr;  
pwidth = MM_2_U1440(125);  
maxpdepth = MM_2_U1440(25);  
AFPBgnPgraph(PageHandle, pwidth, JUSTIFY_ALIGN, DEFAULT_LSP, maxpdepth);  
strcpy (newstr, "this is the first line in my paragraph.");  
AFPPutText (PageHandle,  
newstr, tnr12pt,  
&remainingstr);  
AFPEndPgraph (PageHandle);
```

- **For the AFP Toolbox COBOL Interface:**

Begin a paragraph 125mm wide by up to 25mm deep with justified text and place text in it.

```
MOVE "Page 1 of Many" TO AFP-PAGE-NAME.  
PERFORM AFPBPAGE.  
MOVE 1440 TO AFP-X-COORDINATE.  
MOVE 1440 TO AFP-Y-COORDINATE.  
MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.  
MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.  
PERFORM AFPSPOS.  
MOVE TIM12TYPE TO AFP-FONT-ID.  
PERFORM AFPSFONT.  
* Create a paragraph 3 inches wide up to 1 inch deep  
MOVE 4320 TO AFP-PARA-WIDTH.  
MOVE JUSTIFY-ALIGN TO AFP-FORMAT-OPTION.  
MOVE DEFAULT-LSP TO AFP-LINE-SPACING.  
MOVE 1440 TO AFP-PARA-MAXDEPTH.  
PERFORM AFPBPARA.  
MOVE "This is the first line in my paragraph"  
TO AFP-STRING1.  
MOVE LOW-VALUES TO AFP-CHARACTER-STRING.  
STRING AFP-STRING1  
DELIMITED BY SIZE INTO AFP-CHARACTER-STRING.  
CALL "STRING-LENGTH" USING AFP-CHARACTER-STRING,
```

```

        BY CONTENT LENGTH OF AFP-CHARACTER-STRING,
        BY REFERENCE AFP-STRING-LENGTH.
    ADD 1 TO AFP-STRING-LENGTH.
    PERFORM AFPPTXT.

    MOVE "and this should be concatenated with it."
      TO AFP-STRING2.
    MOVE LOW-VALUES TO AFP-CHARACTER-STRING.
    STRING AFP-STRING2
      DELIMITED BY SIZE INTO AFP-CHARACTER-STRING.
    CALL "STRING-LENGTH" USING AFP-CHARACTER-STRING,
      BY CONTENT LENGTH OF AFP-CHARACTER-STRING,
      BY REFERENCE AFP-STRING-LENGTH.
    ADD 1 TO AFP-STRING-LENGTH.

    PERFORM AFPPTXT.
    PERFORM AFPEPARA.

    PERFORM AFPEPAGE.
```

• **For the AFP Toolbox RPG Interface:**

Begin a paragraph 125mm wide by up to 25mm deep with justified text and place text in it.

```

C          Z-ADD      4040      AFPPARWTH
C          MOVE       JUSTFYALN  AFPFMTOPT
C          MOVE       DEFLSP     AFPLINSPC
C          Z-ADD      10000      AFPPARMAX
C          EXSR       AFPBPARAR
*
C      WDS(1)      CAT(P)  WDS(2):0  AFPCRSTR
C                  CAT(P)  WDS(3):0  AFPCRSTR
C                  CAT(P)  X'00':0  AFPCRSTR
C                  EXSR     AFPPTEXTR
*
C                  EXSR     AFPEPARAR
...
**CTDATA WDS
```

The Toolbox lets you produce customized output pages that utilize typographic fonts, shading, graphics, and image. You can focus your customer's attention on important information that might be overlooked.

---

# Begin Row

## Function

Begins a new row in a table.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPBeginRow(  
    TBHANDLE in_tblHandle,  
    uint in_rowid  
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPBROW.  
    CALL "CBLBeginRow"  
    USING  
    BY VALUE  
    AFP-TABLE-HANDLE AFP-ROW-ID  
    RETURNING  
    AFP-RET-CODE.  
  
    MOVE "CBLBeginRow" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

## Input Parameters

### Table Handle

The handle for the associated table, returned from the Begin Table call.

### Row ID

The ID of the row definition, returned from the associated Define Row call.

## Return Codes

- 1      The table object handle is invalid or null. Call Begin Table before issuing this call.
- 26     There are two possible causes for this return code:
  - The row object handle is invalid or null. Issue Define Row before calling Begin Row.
  - The row ID parameter is invalid because it is zero or is not defined. Ensure that you are using or have defined a table row which contains a valid row id. A valid row id is returned when you successfully call Define Row.
- 36     The table width is incorrect. The user-defined total column widths must be less than or equal to the smaller of the page width and 31680 1440ths, 22 inches, or 558.8mm.

## Sample Function Call

Begins a row, using the ID returned from Define Row for formatting. For a complete example, see “Working with Tables” on page 36.

- **For the AFP Toolbox C Library:**

```
AFPBeginRow(hTbl, HdrRowId);
```

- **For the AFP Toolbox COBOL interface:**

```
MOVE ROW1 TO AFP-ROW-ID.  
PERFORM AFPBROW.
```



## Begin Rule

### Function

Begins drawing a solid rule from the current position in a horizontal or vertical direction. The rule is not actually drawn until the corresponding End Rule is issued. The rule must be ended before you end the page. Ensure that the rule you have specified fits on the logical page.

#### Notes:

1. Begin and End Rule functions are used when you want to draw a variable length rule. If you know the rule length, you should use Put Horizontal Rule or Put Vertical Rule instead for efficiency.
2. To provide device-independent formatting, the output data stream produced by AFP Toolbox has a resolution of 1440 ppi. The printer rounds positioning information from 1440ths of an inch to the nearest pel (240, 300, and so forth). This might cause one pel rounding errors that could be evident in rule widths and rule intersections.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPBgnRule(
    TBHANDLE    in_pageHandle,
    DIRECTION   in_direction,
    ushort      in_ruleThickness,
    TBHANDLE*    outp_ruleHandle
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPBRULE.
    CALL "CBLBgnRule"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-DIRECTION
                AFP-RULE-THICKNESS
            BY REFERENCE
                AFP-RULE-HANDLE
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLBgnRule" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

#### For the AFP Toolbox RPG Interface:

```
DAFPBRULE      PR          10I 0 EXTPROC('AFPBgnRule')
D AFPPAGHDL          *    VALUE
D AFPRULDIR          10I 0 VALUE
D AFPRULTHK          10I 0 VALUE
D AFPRULHDL          *
```

### Input Parameters

#### Page handle

The handle for this page. Page handle is returned by the Begin Page call.

#### Rule Direction

Indicates whether the rule is drawn in the horizontal direction. If the length for a horizontal rule is negative, the rule is drawn to the left of the current position; otherwise it is drawn to the right of the current position. If the length for a vertical rule is negative, the rule extends up from the current position; otherwise it is drawn down from the current position.

## Begin Rule

### Rule thickness

Specifies how thick the rule should be.

## Output Parameters

### Rule handle

The handle for the rule. Rule handle is used on the corresponding EndRule function call.

## Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 1 The **outp\_ruleHandle** value is null. You must specify a valid address for this parameter.

## Sample Function Call

- **For the AFP Toolbox C Library:**

This example prints the word "Name:" at the current position and then draws the rule from after "Name:" to horizontal position 200 millimeters.

```
boxrule = MM_2_UI440 (.25); /* rule will be .25mm thick */
strcpy(word, "Name: ");
rc = AFPWriteString(PageHandle, word, DEFAULT_ICS,
                    DEFAULT_VAR, LEFT_ALIGN, " ", FALSE);
AFPBgnRule(PageHandle, INLINE_DIR, boxrule, &firstrule);
AFPHMoveTo(PageHandle, MM_2_UI440(200));
AFPEndRule(PageHandle, &firstrule);
```

- **For the AFP Toolbox COBOL Interface:**

This example prints the word "Name" at the current position then draws a horizontal rule to a horizontal position of six inches.

```
MOVE "Name: " TO AFP-CHARACTER-STRING.
MOVE 6 TO AFP-STRING-LENGTH.
MOVE LEFT-ALIGN TO AFP-ALIGNMENT-OPTION.
MOVE FALS TO AFP-POSITION-OPTION.
PERFORM AFPWRITE.
MOVE INLINE-DIR TO AFP-DIRECTION.
MOVE 25 TO AFP-RULE-THICKNESS.
PERFORM AFPBRULE.
MOVE 8640 TO AFP-X-COORDINATE.
PERFORM AFPHMOVE.
PERFORM AFPERULE.
```

This example starts 7 vertical rules at the current position each spaced .25 inches apart. Each of the rules is 3 inches long.

```
* A complex rule handle example
MOVE 1440 TO AFP-X-COORDINATE.
MOVE 2880 TO AFP-Y-COORDINATE.
MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
PERFORM AFPSPOS.

MOVE 25 TO AFP-RULE-THICKNESS.
MOVE BASELINE-DIR TO AFP-DIRECTION.
PERFORM AFPBRULE.
SET RULEHNDL1 TO AFP-RULE-HANDLE.

MOVE 360 TO AFP-X-COORDINATE.
PERFORM AFPHMOVE.
PERFORM AFPBRULE.
SET RULEHNDL2 TO AFP-RULE-HANDLE.
MOVE 360 TO AFP-X-COORDINATE.
```

```

PERFORM AFPHMOVE.
PERFORM AFPBRULE.
    SET RULEHNDL3 TO AFP-RULE-HANDLE.
    MOVE 360 TO AFP-X-COORDINATE.
PERFORM AFPHMOVE.
PERFORM AFPBRULE.
    SET RULEHNDL4 TO AFP-RULE-HANDLE.
    MOVE 360 TO AFP-X-COORDINATE.
PERFORM AFPHMOVE.
PERFORM AFPBRULE.
    SET RULEHNDL5 TO AFP-RULE-HANDLE.
    MOVE 360 TO AFP-X-COORDINATE.
PERFORM AFPHMOVE.
PERFORM AFPBRULE.
    SET RULEHNDL6 TO AFP-RULE-HANDLE.
    MOVE 360 TO AFP-X-COORDINATE.
PERFORM AFPHMOVE.
PERFORM AFPBRULE.
    SET RULEHNDL7 TO AFP-RULE-HANDLE.

```

```

* Move down the page 3 inches and turn off the rules
    MOVE 4320 TO AFP-X-COORDINATE.
PERFORM AFPVMOVE.

```

```

    SET AFP-RULE-HANDLE TO RULEHNDL1.
PERFORM AFPERULE.
    SET AFP-RULE-HANDLE TO RULEHNDL2.
PERFORM AFPERULE.
    SET AFP-RULE-HANDLE TO RULEHNDL3.
PERFORM AFPERULE.
    SET AFP-RULE-HANDLE TO RULEHNDL4.
PERFORM AFPERULE.
    SET AFP-RULE-HANDLE TO RULEHNDL5.
PERFORM AFPERULE.
    SET AFP-RULE-HANDLE TO RULEHNDL6.
PERFORM AFPERULE.
    SET AFP-RULE-HANDLE TO RULEHNDL7.
PERFORM AFPERULE.

```

- **For the AFP Toolbox RPG Interface:**

This example prints the word "Name" at the current position then draws a horizontal rule to a horizontal position of eight inches.

```

*Print the word "Name"
*and draw a horizontal rule to a position of 8 inches.
*Specify font for subsequent text data.
C          Z-ADD      TIM14TYPE      AFPFNTID
C          EXSR        AFPSFONTR
*Advance one line in the current font.
C          EXSR        AFPNLINER
*Advance one line in the current font.
C          EXSR        AFPNLINER
*Print the word "Name"
C      'Name:'      CAT(P)      X'00':0      AFPCHRSTR
C          Z-ADD      DEFWS      AFPXTRVAR
C          Z-ADD      DEFICS      AFPXTRICS
C          Z-ADD      LEFTALGN      AFPALNOPT
C          Z-ADD      FALS      AFPPOSOPT
C          MOVE      '.'      AFPALNCHR
C          EXSR        AFPWRITER
*Draw rule to a position of 8 inches.
C          Z-ADD      INLINEDIR      AFPRULDIR
C          Z-ADD      30      AFPRULTHK
C          EXSR        AFPBRULER
*Move to position of 8 inches.
C          Z-ADD      11520      AFPXCRD
C          EXSR        AFPHMOVTOR

```

## Begin Session

---

## Begin Session

### Function

Initializes the AFP Toolbox session. You cannot issue any other AFP Toolbox procedure call until you have successfully executed Begin Session.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPBgnSession(  
    TBHANDLE*    outp_sessHandle  
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPINIT.  
    CALL "CBLBgnSession"  
        USING  
            BY REFERENCE  
            AFPAPI-HANDLE  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLBgnSession" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

#### For the AFP Toolbox RPG Interface:

```
DAFPINTR          PR          10I 0 EXTPROC('AFPBgnSession')  
D  AFPAPIHDL          *
```

### Output Parameters

#### Session handle

The handle for this AFP Toolbox session. This handle is used as input on the Begin Document AFP Toolbox call.

### Return Codes

- 1 The **outp\_sessHandle** value is null. You must specify a valid address for this parameter.

### Sample Function Call

- **For the AFP Toolbox C Library:**

This example begins an AFP Toolbox session.

```
Handle SessHandle;  
rc = AFPBgnSession(&SessHandle);
```

- **For the AFP Toolbox COBOL Interface:**

This example begins an AFP Toolbox session.

```
PERFORM AFPINIT.
```

- **For the AFP Toolbox RPG Interface:**

This example begins an AFP Toolbox session.

```
C          EXSR          AFPINTR
```

## Begin Shade

### Function

Begins a shaded rectangle of specified width at the current location. The rectangle depth is specified by Vertical Move before issuing End Shade. Shading is not actually started until the corresponding End Shade is issued.

**Note:** Use of this function requires microcode support in your printer.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPBgnShade(
    TBHANDLE    in_pageHandle,
    ushort      in_shade_width,
    ushort      in_shade_percent,
    TBHANDLE*    outp_shadeHandle
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPBSHD.
    CALL "CBLBgnShade"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-SHADE-WIDTH
                AFP-SHADE-PERCENT
            BY REFERENCE
                AFP-SHADE-HANDLE
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLBgnShade" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### Page handle

The handle for this page. Page handle is returned by the Begin Shade call.

#### Shade width

The width of the shaded area.

#### Shade percent

Specifies how much shading is applied to the shaded area. The valid values are integers from 0 to 100.

### Return Codes

- 1 The **outp\_shadeHandle** value is null. You must specify a valid address for this parameter.
- 4 The page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

### Sample Function Call

Create a shaded rectangle three inches wide shaded 25 percent:

- **For the AFP Toolbox C library**

## Begin Shade

```
AFPBgnShade (PageHandle,4320,25,&ShadeHandle);
AFPNextLine (PageHandle);
AFPWriteString (PageHandle,"This box is shaded 25 percent",
                DEFAULT_WS,DEFAULT_ICS,LEFT_ALIGN,' ',TRUE) ;
AFPNextLine (PageHandle);
AFPEndShade (PageHandle,&ShadeHandle);
```

- **For the AFP Toolbox COBOL Interface:**

```
        MOVE 4320 TO AFP-SHADE-WIDTH.
        MOVE 25 TO AFP-SHADE-PERCENT.
PERFORM AFPBSHD.
        MOVE 2880 TO AFP-Y-COORDINATE.
PERFORM AFPVMOVE.
PERFORM AFPESHD.
```

## Begin Table

### Function

Begins a table at the current position. After beginning a table, you can format data into rows consisting of fields that can be framed and shaded. See *Begin Field*, *Begin Row*, *Define Field*, *Define Row*, *End Field*, *End Row*, and *End Table* for more information. Also see “Working with Tables” on page 36 for more information about creating tables. The current position at the end of a table is at the bottom-left corner of the table.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPBeginTable(
    TBHANDLE in_pageHandle,
    long in_tblwidth,
    long in_maxtbldepth,
    uint in_tblrotation,
    long in_topthick,
    long in_botthick,
    long in_leftthick,
    long in_rightthick,
    TBHANDLE* out_tblHandle
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPBTL.
    CALL "CBLBeginTable"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-TABLE-WIDTH
                AFP-MAX-TABLE-DEPTH
                AFP-TABLE-ROTATION
                AFP-TOP-THICKNESS
                AFP-BOTTOM-THICKNESS
                AFP-LEFT-THICKNESS
                AFP-RIGHT-THICKNESS
            BY REFERENCE
                AFP-TABLE-HANDLE
            RETURNING
                AFP-RET-CODE.

    MOVE "CBLBeginTable" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

## Input Parameters

### Page Handle

The handle for the current page, returned from the *Begin Page* call.

### Table Width

The width of the table in the current unit of measure. The width of the table should be the sum of the widths of the columns. If the sum of the column widths does not equal the table width, the column widths override the table width. If the table is framed, the left and right vertical rule thickness used for the table frame are included in the width of the table.

The table width must be greater than zero and less than or equal to the smaller of the defined page width or 31680 1440ths, 22 inches, or 558.8mm. To change the page width, use *Set Media Size*.

### Maximum Table Depth

The maximum depth of the table in the current unit of measure. If the table is framed, the thickness of

## Begin Table

the top and bottom horizontal rule of the table frame is included in the depth of the table. If data placed in the table exceeds the table depth, the End Row call that placed the data generates a non-zero return code.

The table depth must be greater than zero and less than or equal to the page length. To change the page length, use Set Media Size.

## Table Rotation

The rotation of the table, clockwise around the table origin. For 0± rotation, the top of the table is parallel to the top of the page or area containing the table. Currently, the only supported value is 0:

### ROTATE0

0± rotation.

## Top Horizontal Rule Thickness

The thickness of the top horizontal rule of the table frame in the current unit of measure. A value of 0 specifies no rule. The thickness must be greater than or equal to zero.

## Bottom Horizontal Rule Thickness

The thickness of the bottom horizontal rule of the table frame in the current unit of measure. A value of 0 specifies no rule. The thickness must be greater than or equal to zero.

## Left Vertical Rule Thickness

The thickness of the left vertical rule of the table frame in the current unit of measure. A value of 0 specifies no rule; however, a value of 0 does not override a field rule. The thickness must be greater than or equal to zero.

## Right Vertical Rule Thickness

The thickness of the right vertical rule of the table frame in the current unit of measure. A value of 0 specifies no rule; however, a value of 0 does not override a field rule. The thickness must be greater than or equal to zero.

## Output Parameters

### Table Handle

The handle for the table. This is required on some subsequent procedure calls, such as Begin Row.

## Return Codes

- 4 The page handle is not valid. Call Begin Page before issuing this call.
- 23 The rule thickness is not valid. Rule Thickness must be greater than or equal to zero.
- 27 There are too many columns specified. You cannot specify more than 27 columns.
- 36 The table width is incorrect. The user-defined total column widths must be less than or equal to the smaller of these values: the page width and 31680 1440ths, 22 inches, or 558.8 millimeters.
- 37 The table depth is not valid. The table depth cannot be greater than MAXYPAGESZ (15000 1440ths, 10.41 inches, or 264.6 millimeters).
- 38 The table rotation is not valid. See the Table Rotation parameter for valid values.
- 42 The row handle is not valid. Call Define Row before issuing this call.
- 43 The field handle is not valid. Call Define Field before issuing this call.
- 61 The table width value is greater than the page width value. Page Width must be greater than Table Width.

## Sample Function Call

- For the AFP Toolbox C Library:



## Begin Table

This command begins a table 125 millimeters wide, at most eight inches deep, and with rules 1 mm thick on top and .5 mm thick everywhere else.

```
RetCode = AFPBeginTable(hPage, MM_2_U1440(125.0),  
    IN_2_U1440(8.0), 0, MM_2_U1440(1), MM_2_U1440(.5),  
    MM_2_U1440(.5), MM_2_U1440(.5), &hTbl);
```

- **For the AFP Toolbox COBOL interface:**

This command begins a table 125mm wide, at most 200 mm (eight inches) deep, with rules 1 mm thick on top and .5 mm thick everywhere else.

```
    MOVE MM TO AFP-UNIT-OF-MEASURE.  
PERFORM AFPSUNI.  
    MOVE 200.0 TO AFP-MAX-TABLE-DEPTH  
    MOVE 125.0 TO AFP-TABLE-WIDTH.  
    MOVE 1.0 TO AFP-TOP-THICKNESS.  
    MOVE .5 TO AFP-BOTTOM-THICKNESS.  
    MOVE .5 TO AFP-LEFT-THICKNESS.  
    MOVE .5 TO AFP-RIGHT-THICKNESS.  
PERFORM AFPBTBL.
```

---

# Begin Underscore

## Function

The begin underscore function starts underscoring text on the page and also specifies whether or not blanks are to be underscored. This call can be issued in the page or inside a paragraph, but not in a table field.

**Note:** The underscoring is actually drawn by the printer so this function requires printer microcode support.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPBgnUScore(  
    TBHANDLE in_pageHandle,  
    boolean in_blankflag  
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPBUSC.  
  CALL "CBLBgnUScore"  
  USING BY VALUE  
    AFP-PAGE-HANDLE,  
    AFP-US-BLANKS,  
  RETURNING  
    AFP-RET-CODE.
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Blank Flag

A boolean value that specifies whether the blanks in the string of text are undercored (true) or not (false). Valid values are:

**TRUE (C), TRU (COBOL)**

Blanks in the string of text are underscored.

**FALSE (C), FALS (COBOL)**

Blanks in the string of text are not underscored.

## Return Codes

**14** This function is not allowed inside a table. Issue End Table before issuing this call.

## Sample Function Call

### • For the AFP Toolbox C Library:

Underline a string of text, including the blanks within that string.

```
rc = AFPBgnUScore (PageHandle, TRUE);  
rc = AFPWriteString (PageHandle, AFP_STRING_1,  
  DEFAULT_WS,DEFAULT_ICS,LEFT_ALIGN,' T,FALSE) ;  
rc = AFPEndUScore (PageHandle);
```

### • For the AFP Toolbox COBOL Interface:

Underline a string of text, including the blanks within that string.

```
PERFORM AFPBPARA.  
  MOVE LOW-VALUES TO AFP-CHARACTER-STRING  
  STRING AFP-STRING-1
```

```
        DELIMITED BY SIZE INTO AFP-CHARACTER-STRING
CALL "STRING-LENGTH" USING AFP-CHARACTER-STRING,
        BY CONTENT LENGTH OF AFP-CHARACTER-STRING,
        BY REFERENCE AFP-STRING-LENGTH
PERFORM AFPPTTEXT.
        MOVE TRU TO AFP-US-BLANKS
PERFORM AFPBUSC.
        MOVE LOW-VALUES TO AFP-CHARACTER-STRING
        STRING AFP-STRING-2
        DELIMITED BY SIZE INTO AFP-CHARACTER-STRING
CALL "STRING-LENGTH" USING AFP-CHARACTER-STRING,
        BY CONTENT LENGTH OF AFP-CHARACTER-STRING,
        BY REFERENCE AFP-STRING-LENGTH
PERFORM AFPPTTEXT.
PERFORM AFPEUSC.
PERFORM AFPEPARA.
```

---

# Create Link

## Function

Creates a link area at the current location to the specified target. Links are used for navigating within the document or to external documents or programs, using programs such as AFP Workbench or OnDemand. The rectangle defined by the link width and depth is the area on the screen (when using Workbench) that becomes the source link.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPCreateLink(
    TBHANDLE    in_pageHandle,
    ulong       in_link_width,
    ulong       in_link_depth,
    boolean     in_is_executable,
    char*       in_target_name,
    char*       in_target_parms
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPLINK.
    CALL "CBLCreateLink"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-LINK-WIDTH
                AFP-LINK-DEPTH
                AFP-IS-EXECUTABLE
            BY CONTENT
                AFP-TARGET-NAME
                AFP-TARGET-PARMS
        RETURNING
            AFP-RET-CODE.
    MOVE "AFPCreateLink" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPLINK      PR          10I 0 EXTPROC('AFPCreateLink')
D AFPPAGHDL          *  VALUE
D AFPLNKWID          10I 0  VALUE
D AFPLNKDEP          10I 0  VALUE
D AFPISEXEC          10I 0  VALUE
D AFPTGTNAM          250
D AFPTGTPRM          250
```

## Input Parameters

### Page handle

Current page handle returned by Begin Page.

### Link width

The width of the rectangular space that is defined for this link. The upper left corner of this rectangular space is at the current position.

### Link depth

The depth of the rectangular space that is defined for this link. The upper left corner of this rectangular space is at the current position.

**Is executable**

Specifies whether the target is an executable program (if TRUE) or a document (if FALSE). See Target name description below for more information.

**Target name**

The name of the object being linked to. This object can be an external document name or an executable program. If the link name is a document, when the link is selected by a user of the AFP Workbench, another instance of the Workbench is invoked to process the document. The document name can be up to 250 characters in length. If more than 250 bytes are specified, the name is truncated. If the link name is executable, then that program is invoked and any parameters specified with this call are passed to the program.

**Target parameters**

A string of up to 250 bytes passed as input to the executable program specified as the link name, when called by the user of the AFP Workbench. If more than 250 bytes are specified, the string is truncated.

**Return Codes**

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

**Sample Function Call**

- **For the AFP Toolbox C Library:**

Build a link to program “applic1.exe”. The link appears at the current location and is one inch wide and two inches deep. When a user of the AFP Workbench looks at the output from this application and clicks on this link area, the Workbench invokes “applic1.exe” with parameters “1163234 988324 Jane Doe”.

```
strcpy(TargetName, " applic1.exe");
strcpy(TargetParms, "1163234 988324 Jane Doe");
rc = AFPCreateLink(PageHandle, 1440, 2880, TRUE, TargetName, TargetParms);
```

- **For the AFP Toolbox COBOL Interface:**

Build a link to program “applic1.exe”. The link appears at the current location and is one inch wide and two inches deep. When a user of the AFP Workbench looks at the output from this application and clicks on this link area, the Workbench invokes “applic1.exe” with parameters “1163234 988324 Jane Doe”.

```
* BgnPage 1
  PERFORM AFPBPAGE.
* SetPos
  MOVE 1440 TO AFP-X-COORDINATE.
  MOVE 1440 TO AFP-Y-COORDINATE.
  MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
  MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
  PERFORM AFPSPOS.
* Stick the CreateLink xmp in here
  MOVE "applic1.exe" TO AFP-TARGET-NAME.
  MOVE 1440 TO AFP-LINK-WIDTH.
  MOVE 2880 TO AFP-LINK-DEPTH.
  MOVE TRU TO AFP-IS-EXECUTABLE.
  MOVE "1163234 988324 Jane Doe"
    TO AFP-TARGET-PARMS.
  PERFORM AFPCLINK.
* EndPage
  PERFORM AFPEPAGE.
```

- **For the AFP Toolbox RPG Interface:**

## Create Link

Build a link to program “applic1.exe”. The link appears at the current location and is one inch wide and two inches deep. When a user of the AFP Workbench looks at the output from this application and clicks on this link area, the Workbench invokes “applic1.exe” with parameters “1163234 988324 Jane Doe”.

C	MOVEL	'LinkName'	AFPTGTNAM
C	MOVEL	'LinkParms'	AFPTGTPRM
C	Z-ADD	1440	AFPLNKWID
C	Z-ADD	1440	AFPLNKDEP
C	EXSR	AFPCLINKR	

## Data Matrix Bar Code Data

### Function

Adds data to a Data Matrix bar code object previously started with Begin Bar Code.

### Syntax

```
int AFPDataMatrix (TBHANDLE      in_bcocaHandle,
                   ushort         in_bcsymxpos,
                   ushort         in_bcsymypos,
                   short          in_bcnumrows,
                   short          in_bcrowsize,
                   boolean        in_bce2a,
                   boolean        in_bcescape,
                   short          in_bcseqcount,
                   short          in_bcseqind,
                   byte           in_bcspecfunc,
                   byte           in_bcfileid1,
                   byte           in_bcfileid2,
                   uchar*         in_bcddata);
```

Figure 9. C syntax

```
AFPDMTRX.
    CALL "CBLDataMatrix"
        USING
        BY VALUE
            AFP-BCOCA-HANDLE
            AFP-BCSYM-XPOS,
            AFP-BCSYM-YPOS,
            AFP-BC-NUMROWS
            AFP-BC-ROWSIZE
            AFP-BC-E2A
            AFP-BC-ESCAPE
            AFP-BC-SEQCOUNT
            AFP-BC-SEQIND
            AFP-BC-SPECFUNC
            AFP-BC-FILEID1
            AFP-BC-FILEID2
        BY CONTENT
            AFP-BC-DATA
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLDataMatrix" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

Figure 10. COBOL syntax

### Input Parameters

#### BCOCA Handle

The handle for the bar code object to contain this data. BCOCA handle is returned by the Begin Bar Code call.

#### Symbol X and Y position

Specifies the X and Y offsets from the presentation space origin to the top left corner of the beginning of the bar code symbol. Valid range is from 1 to 32767 1440ths. Zero is NOT valid for X or Y.

#### Number of Rows

The desired number of rows in the generated bar code symbol, including the finder pattern. Specify 0

## Data Matrix Bar Code Data

as the number of rows to have the printer generate the minimum number of rows necessary. The number of rows must be an even number from 8 to 144, but not all numbers are allowed. The supported sizes for Data Matrix bar codes vary depending on the number of rows, row size, and module size. See the *Bar Code Object Content Architecture Reference* for details.

### Row Size

The number of modules in each row including the finder pattern. There must be an even number of modules per row and an even number of rows. There are square symbols with sizes from 10x10 to 144x144 and rectangular sizes from 8x18 to 16x48, not including quiet zones.

### EBCDIC to ASCII translation

This flag indicates whether the bar code data should be converted from EBCDIC to ASCII encoding. In order to print correctly the data and macro must be encoded in ASCII code page 819. In general, if you are running on a host platform such as OS/390 or AS/400, your data is EBCDIC by default so you should specify “true” for this parameter. On a workstation platform you would normally specify “false”.

### Ignore Escape Sequence

This flag indicates whether a backslash character (ASCII X'5C' or EBCDIC X'E0') within the bar code data should be treated as an escape sequence or not. Specify “false” if the backslash should be treated as an escape, and “true” if backslashes should be treated as character data.

### Sequence Count and Indicator

Multiple Data Matrix bar code symbols can be logically linked together to encode large amounts of data. This is called a structured append sequence. The logically linked symbols can be printed separately and then logically recombined after they are scanned. From 2 to 16 Data Matrix symbols can be linked together. The Sequence Count specifies the number of symbols to be linked. The Sequence Indicator specifies for each bar code symbol where it fits logically into the sequence (a number from 1 to 16). The Sequence Indicator must be 1 if the Sequence Count is 1.

### Special Functions

This parameter is used to request special functions that can be used with a Data Matrix symbol. Valid values are:

<b>FCN1UCC</b>	UCC/EAN FNC1 alternate data type identifier. An FNC1 will be added to the output data when the bar code is printed, indicating that this bar code symbol conforms to the UCC/EAC application identifier standard format.
<b>FCN1IND</b>	Industry FNC1 alternate data type identifier. An FNC1 will be added to the output data when the bar code is printed, indicating that this bar code symbol conforms to a particular industry standard format.
<b>RDRPROG</b>	Reader Programming. The bar code data contains a message used to program the bar code reader. If RDRPROG is specified then the Sequence Count must be zero (structured appends are not supported).
<b>MAC5 or MAC6</b>	Header and Trailer Macros. These values instruct the bar code reader to insert an industry specific header and trailer around the symbol data. The bar code data will contain a 05 Macro or 06 Macro codeword. If MAC5 or MAC6 is specified then the Sequence Count must be zero (structured appends are not supported).
<b>USERDEF</b>	User Default. No special function processing is to be done on the bar code symbol.

### File ID Bytes

These two parameters specify the high- and low-order bytes of a unique two byte file identification for a set of structured append symbols. This helps ensure that the symbols from two different structured appends are not linked together. You should specify the same File ID bytes for any symbols that are to be linked together. These parameters are ignored if the Sequence Count is 0 or 1.

### Data

The string of characters to be encoded as bar code data. For Data Matrix bar codes, up to 3116 bytes



depending on whether the data is character or numeric. For details refer to the symbology specification. The input data can be ASCII or EBCDIC; use the E2A parameter to indicate whether translation from EBCDIC codepage 500 to ASCII codepage 819 is required.

## Return Codes

- 1** The **outp\_bcHandle** value is null. You must specify a valid address for this parameter.
- 100** The bar code position is out of range. The valid range is from 1—32767.
- 101** The number of rows must be 0 or an even number in the range 10—144.  
The number of data symbols per row must be 0 or an even number in the range 10—144.
- 103** The sequence count must be in the range 0—16.  
The sequence indicator must be in the range 0—16.
- 104** The fileID bytes must be in the range X'01'—X'FE'.

## Sample Function Call

Create a Data Matrix bar code:

### C Programming Language

```
rc = AFPBgnBarCode(hPage, "BCOCA 1", MM_2_U1440(100),
  MM_2_U1440(100), DEG_0, DATAMATRIX_BC, 0x00, DEFAULT_BCFONT,
  BLACK, DEFAULT_BCMODWIDTH, MM_2_U1440(24), -1, -1,
  &hBcoca);

rc = AFPDataMatrix(hBcoca, 1, 1, 0, 10, TRUE, FALSE, 0, 1, USERDEF,
  0x01, 0x01, "1234567");

rc = AFPEndBarCode(hPage, &hBcoca);
```

### COBOL Programming Language

```
MOVE "DATAMATRIX BARCODE" TO AFP-BCOCA-NAME.
MOVE 100 TO AFP-AREA-XSIZE.
MOVE 100 TO AFP-AREA-YSIZE.
MOVE DATAMATRIX-BC TO AFP-BC-TYPE.
MOVE 0 TO AFP-BC-MODIFIER.
MOVE -1 TO AFP-BC-FONT-ID.
MOVE DEG-0 TO AFP-OBJECT-ROTATION.
PERFORM AFPBBBCD.
MOVE 1 TO AFP-BCSYM-XPOS.
MOVE 1 TO AFP-BCSYM-YPOS.
MOVE FALS TO AFP-BCHRI-PRESENT.
MOVE 0 TO AFP-BC-NUMROWS.
MOVE 10 TO AFP-BC-ROWSIZE.
MOVE TRU TO AFP-BC-E2A.
MOVE FALS TO AFP-BC-ESCAPE.
MOVE 0 TO AFP-BC-SEQCOUNT.
MOVE 1 TO AFP-BC-SEQIND.
MOVE RDRPROG TO AFP-BC-SPECFUNC.
MOVE 5 TO AFP-BC-FILEID1.
MOVE 100 TO AFP-BC-FILEID2.
MOVE "the quick brown fox jumped over the lazy dog"
TO AFP-BC-2DDATA.
PERFORM AFPMXBC.
PERFORM AFPEBCD.
```

---

# Define Double-Byte Font By Attribute

## Function

Finds a font on your system that matches specified character set and code page combinations and returns a font ID. The code page information is used to translate a DBCS string into the font's code set. The function finds only an outline font. An AFP SBCS code page and AFP DBCS code page must be specified. The font ID is used on subsequent Set Font calls.

**Note:** Use of the API requires the access to AFP DBCS outline fonts. (AFP DBCS raster fonts are not supported.) Refer to the AFP Font Collection products for information on how to build DBCS outline fonts. This is the only call for defining a DBCS font.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPDefineDbFontByAttr(
    char*      in_SBCS_codePage,
    char*      in_DBCS_codePage,
    char*      in_codeset,
    char*      in_descriptiveName,
    ushort     in_pointSize,
    ushort     in_horizontal_scale_factor
    FDSWeights in_weight,
    FDSWidths  in_width,
    FontStyle  in_style,
    FontID*    out_fontID
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPDFNTDB.
  CALL      "CBLDefineDbFontByAttr"
    USING
      BY CONTENT
        AFP-CODE-PAGE
        AFP-DBCODE-PAGE
        AFP-CODE-SET
        AFP-DESCRIPTIVE-NAME
      BY VALUE
        AFP-POINT-SIZE
        AFP-HSCALE
        AFP-WEIGHT
        AFP-FONT-WIDTH
        AFP-STYLE
      BY REFERENCE
        AFP-FONT-ID
      RETURNING
        AFP-RET-CODE.
  MOVE "CBLDefineDbFontAttr" TO AFP-ERRDATA.
  PERFORM CHKSUCC.
```

## Input Parameters

### SBCS code page

The SBCS code page name of the output data (for example, T1H00290).

### DBCS code page

The DBCS code page name of the output data (for example, T10300).

### Codeset

The code set name of the output data (for example, "IBM-939")

### Descriptive name

Name of the font (for example, "HeiseiMincho"). The value for the descriptive name parameter is case sensitive.

### Point size

Size of the font specified in decipoints (point size multiplied by 10). For example, a 12 point font would be specified as 120.

### Horizontal scaling factor

A horizontal scaling factor expressed as a percentage of the point size. For example, if you choose a value of 50, the width of your font will be 50% of the height. No scaling is performed if values of either 0 or 100 are entered.

### Weight

The thickness of the font, for example, medium or bold. Valid values are:

ULTRA\_LIGHT\_WT (for C) or ULTRALIGHT (for COBOL)  
 EXTRA\_LIGHT\_WT (for C) or EXTRALIGHT (for COBOL)  
 LIGHT\_WT (for C) or LIGHT (for COBOL)  
 SEMI\_LIGHT\_WT (for C) or SEMILIGHT (for COBOL)  
 MEDIUM\_WT (for C) or MEDIUM (for COBOL)  
 SEMI\_BOLD\_WT (for C) or SEMIBOLD (for COBOL)  
 BOLD\_WT (for C) or BOLD (for COBOL)  
 EXTRA\_BOLD\_WT (for C) or EXTRABOLD (for COBOL)  
 ULTRA\_BOLD\_WT (for C) or ULTRABOLD (for COBOL)

### Width

The width of the font, for example normal or condensed. Valid values are:

ULTRA\_CONDENSED\_WD (for C) or ULTRACOND (for COBOL)  
 EXTRA\_CONDENSED\_WD (for C) or EXTRACOND (for COBOL)  
 CONDENSED\_WD (for C) or CONDENSED (for COBOL)  
 SEMI\_CONDENSED\_WD (for C) or SEMICONDENSED (for COBOL)  
 MEDIUM\_WD (for C) or NORML (for COBOL)  
 SEMI\_EXPANDED\_WD (for C) or SEMIEXP (for COBOL)  
 EXPANDED\_WD (for C) or EXPANDED (for COBOL)  
 EXTRA\_EXPANDED\_WD (for C) or EXTRAEXP (for COBOL)  
 ULTRA\_EXPANDED\_WD (for C) or ULTRAEXP (for COBOL)

### Style

The style of the font, either roman or italic. Valid values are:

NORMAL\_FS (roman) (for C) or ROMAN (for COBOL )  
 ITALIC\_FS (for C) or ITALIC (for COBOL)

## Output Parameters

### Font ID

The font ID for use on subsequent PText::SetFont calls.

## Return Codes

See "Font Return Codes" on page 259 for a listing of the return codes for this call.

## Sample Function Call

.

### For the AFP Toolbox C Library:

This example defines a 14 point Heisei Gothic font.

```
rc = AFPDefineDbFontByAttr (
    "T1H01027",          // SBCS codepage
    "T10300",            // DBCS codepage
```

## Define Double-Byte Font By Attribute

```
"IBM-939",           // Codeset
"HeiseiKakuGothic",  // Font name
140,                 // Deci-point size
0,                  // Scale factor
MEDIUM_WT,          // Weight
MEDIUM_WD,           // Width
NORMAL_FS,           // Font style
&FontID              // Font handle
);
```

- **For the AFP Toolbox COBOL Interface:**

To define a Times New Roman 12 point, bold italic font:

```
MOVE "T1H01027" TO AFP-CODE-PAGE.
MOVE "T10300" TO AFP-DBCODE-PAGE.
MOVE "IBM-939" TO AFP-CODE-SET
MOVE "HeiseiKakuGothic" TO AFP-DESCRIPTIVE-NAME.
MOVE 140 TO AFP-POINT-SIZE.
MOVE 0 TO AFP-HSCALE.
MOVE MEDIUM TO AFP-WEIGHT.
MOVE MEDIUM TO AFP-FONT-WIDTH.
MOVE ROMAN TO AFP-STYLE.
PERFORM AFPDFNTDB.
MOVE AFP-FONT-ID TO TNR12.
```

## Define Field

### Function

Creates a field definition for a table. Subsequent Begin Field procedure calls use the field definition. You can use Put Text in Table Field to put data into the field.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPDefineField(
    TBHANDLE in_docHandle,
    AlignmentOption in_format,
    long in_alignpos,
    VertAlignOption in_vertform,
    long in_leftmargin,
    long in_rightmargin,
    long in_linespacing,
    TextOrientation in_txtorient,
    uint in_shadingintensity,
    short far* out_fieldid
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPDFLD.
    CALL "CBLDefineField"
    USING
        BY VALUE
            AFP-DOCUMENT-HANDLE
            AFP-FORMAT-OPTION
            AFP-ALIGNMENT-POSITION
            AFP-VERTICAL-FORMAT
            AFP-LEFT-MARGIN
            AFP-RIGHT-MARGIN
            AFP-LINE-SPACING
            AFP-TEXT-ORIENTATION
            AFP-SHADE-PERCENT
        BY REFERENCE
            AFP-FIELD-ID
    RETURNING
        AFP-RET-CODE
    MOVE "CBLDefineField" TO AFP-ERRDATA.
    PERFORM CHKSCC.
```

## Input Parameters

### Document Handle

The handle for the current document returned from the Begin Document call.

### Format Option

The type of formatting to be performed on formatted text placed with the Put Text procedure call in the field:

#### **LEFT\_ALIGN for C, or F0LEFT for COBOL**

Text is aligned at the left margin of the field, with a ragged right margin.

#### **CENTER\_ALIGN for C, or F0CENTER for COBOL**

Text is centered between the left and right margins of the field.

#### **RIGHT\_ALIGN for C, or F0RIGHT for COBOL**

Text is aligned at the right margin of the field.

## Define Field

### **JUSTIFY for C, or F0JUSTIFY for COBOL**

Text is aligned at both the left margin and right margin of the field. The formatter varies the space between words to stretch or shrink a line to fit.

### **Alignment Position**

The position within the field for character alignment. The position is specified as an offset from the left margin of the field. The value must be greater than or equal to -1 and less than the width of the field.

This parameter is used only when placing data with the Put Character String call with Character Alignment Option.

### **Vertical Format**

The vertical alignment option for lines of text within the field. The valid values are:

#### **VERTOP for C, or VERTOP for COBOL**

Align the text at the top of the field.

#### **VERCENTER for C, or VERCENTER for COBOL**

Align the text in the center of the field.

#### **VERBOTTOM for C, or VERBOTTOM for COBOL**

Align the text at the bottom of the field.

### **Left Margin**

The left margin for lines of text in the field as an offset from the left edge of the left vertical rule.

### **Right Margin**

The right margin for lines of text in the field as an offset from the left edge of the right vertical rule.

### **Line Spacing**

The spacing between subsequent lines of text in the current unit of measure. You can specify a specific value or use the default baseline increment of the font. If the default is specified, the line spacing associated with the largest font in each line of text is used.

This parameter has no effect on unformatted data; that is, on data included with the Put Character String call. You can specify a value greater than or equal to -1, or use the default line spacing. To use the default, specify DEFAULT\_LSP for C, or DEFAULT-LSP for COBOL.

### **Text Orientation**

The degree of orientation for lines of text in the field. Currently, the only supported value is 0:

#### **0 for C TXT0R0-0 for COBOL**

0, 0 text orientation

### **Shading Intensity**

A value between 0-100 that specifies the shading intensity to be used, with 1 being the least intense. 0 indicates no shading. The shading intensity output depends on the printer. Shading requires printer microcode support.

### **Top Horizontal Rule Thickness**

The thickness of the top horizontal rule. A value of 0 specifies no rule.

### **Bottom Horizontal Rule Thickness**

The thickness of the bottom horizontal rule. A value of 0 specifies no rule.

### **Left Vertical Rule Thickness**

The thickness of the left vertical rule. A value of 0 specifies no rule.

### **Right Vertical Rule Thickness**

The thickness of the right vertical rule. A value of 0 specifies no rule.

## Output Parameters

### **Field ID**

The field ID for use on associated Begin Field calls.

## Return Codes

- 3** The document handle is missing or is not valid. Make sure you have called Begin Document before issuing this call.
- 20** An error occurred allocating the table field object. This may be a result of insufficient system memory.
- 29** The table format option is not correct. See the Format Option parameter for valid values.
- 30** The table field text orientation is not valid. See the Text Orientation parameter for valid values.
- 31** The alignment position is not valid. See the Alignment Position parameter for valid values.
- 32** The table margin is not valid. The margin must be greater than or equal to 0.
- 33** The line spacing is not valid. See the Line Spacing parameter for valid values.
- 54** The shading intensity is not valid. The value must be between 0 and 100.
- 62** The vertical format is not valid. See the Vertical Format parameter for valid values.

## Sample Function Call

- **For the AFP Toolbox C Library:**

This command defines characteristics for a field that uses shading intensity 18, has text that is left aligned and vertically centered, has no left or right margin, and specifies no text rotation.

```
FormatOption = LEFT_ALIGN;
VertFormat   = VERCENTER;
TxtOrient    = IOB90_T0;
AlignPos     = 0;
```

```
LeftMargin   = 0;
RightMargin  = 0;
```

```
LineSpace    = 1;
```

```
ShadeIntense = 18.0;
```

```
AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
               LeftMargin, RightMargin, LineSpace, TxtOrient,
               ShadeIntense, HdrFldArray[Index]);
```

- **For the AFP Toolbox COBOL interface:**

This command defines a field with shading intensity 18, no left or right margins, text centered horizontally and vertically, and rules .5 mm thick.

```
MOVE MM TO AFP-UNIT-OF-MEASURE.
PERFORM AFPSUNI.
MOVE 18 TO AFP-SHADING-INTENSITY.
MOVE 0 TO AFP-ALIGNMENT-POSITION.
MOVE 0.0 TO AFP-LEFT-MARGIN.
MOVE 0.0 TO AFP-RIGHT-MARGIN.
MOVE AFP-DEFAULT TO AFP-LINE-SPACING.
MOVE .5 TO AFP-TOP-THICKNESS.
MOVE .5 TO AFP-BOTTOM-THICKNESS.
MOVE .5 TO AFP-LEFT-THICKNESS.
MOVE .5 TO AFP-RIGHT-THICKNESS.
PERFORM AFPDFLD.
```

---

# Define Font By Attribute

## Function

Finds a font on your system that matches specified attributes and returns a font ID corresponding to this font. The font ID is used on subsequent Set Font calls.

**Note:** Define Font can only *define* an existing font to use, it does not *create* a font with these attributes.

For information about obtaining a list of fonts available on your system, see the appendix for your operating system.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPDefineFontByAttr(
    char*      in_codePage,
    char*      in_descriptiveName,
    ushort     in_pointSize,
    FDSWeights in_weight,
    FDSWidths  in_width,
    FontStyle  in_style,
    FontID*    out_fontID
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPDFNTAT.
    CALL "CBLDefineFontByAttr"
        USING
            BY CONTENT
                AFP-CODE-PAGE
                AFP-DESCRIPTIVE-NAME
            BY VALUE
                AFP-POINT-SIZE
                AFP-WEIGHT
                AFP-FONT-WIDTH
                AFP-STYLE
            BY REFERENCE
                AFP-FONT-ID
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLDefineFontAttr" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPDFNTAT      PR          10I 0  EXTPROC('AFPDefineFontByAttr')
D  AFPCDEPAG          9
D  AFPDESNAM        120
D  AFPPNTSIZ         5I 0  VALUE
D  AFPWGT           10I 0  VALUE
D  AFPFNTWTH        10I 0  VALUE
D  AFPSTL           10I 0  VALUE
D  AFPFNTID         5I 0
```

## Input Parameters

### Code page

The code page name (the member name in the font library, for example, T1V10500) of the output



data. This is not the same as the input code page in which the text data is entered. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, \_ (underscore), #, and @. Code page names must be in uppercase.

The default input code page for all text in AFP Toolbox is assumed to be the multinational ASCII code page T1000850 for workstation systems and EBCDIC T1V10500 for host systems (unless you've changed this value with the Set Input Codepage call).

**Descriptive name**

Name of the font (for example, "TIMES NEW ROMAN LATIN1"). The value for the descriptive name parameter is case sensitive and must exactly match the name of the font typeface as shown in the font listing. All descriptive names in IBM-supplied fonts are usually defined in uppercase.

**Point size**

Size of the font specified in decipoints (point size multiplied by 10). For example, a 12 point font would be specified as 120. For raster fonts, specify a point size you know exists in your font library. For outline fonts, fractional point sizes are allowed. For example, a 12.5 point font would be specified as 125.

**Weight**

The thickness of the font, for example, medium or bold. The valid values are:

- ULTRA\_LIGHT\_WT (for C) or ULTRALIGHT (for COBOL and RPG)
- EXTRA\_LIGHT\_WT (for C) or EXTRALIGHT (for COBOL and RPG)
- LIGHT\_WT (for C) or LIGHT (for COBOL and RPG)
- SEMI\_LIGHT\_WT (for C) or SEMILIGHT (for COBOL and RPG)
- MEDIUM\_WT (for C) or MEDIUM (for COBOL and RPG)
- SEMI\_BOLD\_WT (for C) or SEMIBOLD (for COBOL and RPG)
- BOLD\_WT (for C) or BOLD (for COBOL and RPG)
- EXTRA\_BOLD\_WT (for C) or EXTRABOLD (for COBOL and RPG)
- ULTRA\_BOLD\_WT (for C) or ULTRABOLD (for COBOL and RPG)

**Width**

The width of the font, for example normal or condensed. The valid values are:

- ULTRA\_CONDENSED\_WD (for C) or ULTRACOND (for COBOL and RPG)
- EXTRA\_CONDENSED\_WD (for C) or EXTRACOND (for COBOL and RPG)
- CONDENSED\_WD (for C) or CONDENSED (for COBOL and RPG)
- SEMI\_CONDENSED\_WD (for C) or SEMICONDENSED (for COBOL and RPG)
- MEDIUM\_WD (for C) or NORML (for COBOL and RPG)
- SEMI\_EXPANDED\_WD (for C) or SEMIEXP (for COBOL and RPG)
- EXPANDED\_WD (for C) or EXPANDED (for COBOL and RPG)
- EXTRA\_EXPANDED\_WD (for C) or EXTRAEXP (for COBOL and RPG)
- ULTRA\_EXPANDED\_WD (for C) or ULTRAEXP (for COBOL and RPG)

**Style**

The style of the font, either roman or italic. The valid values are:

- NORMAL\_FS (roman) (for C) or ROMAN (for COBOL and RPG)
- ITALIC\_FS (for C) or ITALIC (for COBOL and RPG)

**Output Parameters****Font ID**

The font ID for use on subsequent Set Font calls.

**Return Codes**

See "Font Return Codes" on page 259 for a listing of the return codes for this call.

## Define Font By Attribute

### Sample Function Call

- **For the AFP Toolbox C Library:**

To define a Times New Roman 12 point, bold italic font:

```
strcpy(Typeface,"TIMES NEW ROMAN LATIN1");
Codepage="T1V10500"; /*EBCDIC code page 500*/
rc = AFPDefineFontByAttributes(Codepage, Typeface, 120, BOLD_WT,
                              MEDIUM_WD, ITALIC_FS, &FontID);
```

- **For the AFP Toolbox COBOL Interface:**

To define a Times New Roman 12 point, bold italic font:

```
MOVE "T1V10500" TO AFP-CODE-PAGE.
MOVE "TIMES NEW ROMAN LATIN1" TO AFP-DESCRIPTIVE-NAME.
MOVE 120 TO AFP-POINT-SIZE.
MOVE MEDIUM TO AFP-WEIGHT.
MOVE NORML TO AFP-FONT-WIDTH.
MOVE ROMAN TO AFP-STYLE.
PERFORM AFPDFNTAT.
MOVE AFP-FONT-ID TO TNR12.
```

- **For the AFP Toolbox RPG Interface:**

To define a Times New Roman 12 point, bold italic font:

```
* Define the fonts
C      DTA(6)      CAT(P)      x'00':0      AFPDESNAM
C      DTA(7)      CAT(P)      x'00':0      AFPCDEPAG
C              Z-ADD      140      AFPPNTSIZ
C              Z-ADD      MEDIUM      AFPWGT
C              Z-ADD      NORMAL      AFPFNTWTH
C              Z-ADD      ROMAN      AFPSTL
C              EXSR      AFPDFNTATR
C              Z-ADD      AFPFNTID      TIM14TYPE
...
**CTDATA DTA
TIMES NEW ROMAN LATIN1      6
T1V10500      7
CON20000      8
```

## Define Font By Attributes With Scaling

### Function

Finds a font on your system that matches specified attributes and returns a font ID corresponding to this font. The font ID is used on subsequent Set Font calls. You must also specify Set Font Type to select outline fonts in order to use this function.

**Note:** Define Font can only *define* an existing font to use, it does not *create* a font with these attributes.

For information about obtaining a list of fonts available on your system, see the appendix for your operating system.

**Note:** You need only use this call if you want to scale your font differently in the horizontal as opposed to the vertical direction. Otherwise, you only need use the Define Font By Attribute function call.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPDefineFontByAttrWithScaling(
    char*      in_codePage,
    char*      in_descriptiveName,
    ushort    in_pointSize,
    ushort    in_hscale,
    FDSWeights in_weight,
    FDSWidths in_width,
    FontStyle  in_style,
    FontID*    out_fontID
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPDEFNTSC.
    CALL "CBLDefineFontByAttrWithScaling"
        USING
            BY CONTENT
                AFP-CODE-PAGE
                AFP-DESCRIPTIVE-NAME
            BY VALUE
                AFP-POINT-SIZE
                AFP-HSCALE
                AFP-WEIGHT
                AFP-FONT-WIDTH
                AFP-STYLE
            BY REFERENCE
                AFP-FONT-ID
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLDefineFontByAttrWithScaling" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### Code page

The code page name (the member name in the font library, for example, T1V10500) of the output data. This is not the same as the input code page in which the text data is entered. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, \_ (underscore), #, and @. Code page names must be in uppercase.

## Define Font By Attributes With Scaling

The default input code page for all text in AFP Toolbox is assumed to be the multinational ASCII code page T1000850 for workstation systems and EBCDIC T1V10500 for host systems (unless you've changed this value with the Set Input Codepage call).

### Descriptive name

Name of the font (for example, "TIMES NEW ROMAN LATIN1"). The value for the descriptive name parameter is case sensitive. All descriptive names in IBM-supplied fonts are usually defined in uppercase.

### Point size

Size of the font specified in decipoints (point size multiplied by 10). For example, a 12 point font would be specified as 120. For raster fonts, specify a point size you know exists in your font library. For outline fonts, fractional point sizes are allowed. For example, a 12.5 point font would be specified as 125.

### H scale

A horizontal scaling factor expressed as a percentage of the point size. For example, if you choose a value of 50, the width of your font will be 50% of the height. No scaling is performed if values of either 0 or 100 are entered.

### Weight

The thickness of the font, for example, medium or bold. The valid values are:

ULTRA\_LIGHT\_WT (for C) or ULTRALIGHT (for COBOL and RPG)  
EXTRA\_LIGHT\_WT (for C) or EXTRALIGHT (for COBOL and RPG)  
LIGHT\_WT (for C) or LIGHT (for COBOL and RPG)  
SEMI\_LIGHT\_WT (for C) or SEMILIGHT (for COBOL and RPG)  
MEDIUM\_WT (for C) or MEDIUM (for COBOL and RPG)  
SEMI\_BOLD\_WT (for C) or SEMIBOLD (for COBOL and RPG)  
BOLD\_WT (for C) or BOLD (for COBOL and RPG)  
EXTRA\_BOLD\_WT (for C) or EXTRABOLD (for COBOL and RPG)  
ULTRA\_BOLD\_WT (for C) or ULTRABOLD (for COBOL and RPG)

### Width

The width of the font, for example normal or condensed. The valid values are:

ULTRA\_CONDENSED\_WD (for C) or ULTRACOND (for COBOL and RPG)  
EXTRA\_CONDENSED\_WD (for C) or EXTRACOND (for COBOL and RPG)  
CONDENSED\_WD (for C) or CONDENSED (for COBOL and RPG)  
SEMI\_CONDENSED\_WD (for C) or SEMICONDENSED (for COBOL and RPG)  
MEDIUM\_WD (for C) or NORML (for COBOL and RPG)  
SEMI\_EXPANDED\_WD (for C) or SEMIEXP (for COBOL and RPG)  
EXPANDED\_WD (for C) or EXPANDED (for COBOL and RPG)  
EXTRA\_EXPANDED\_WD (for C) or EXTRAEXP (for COBOL and RPG)  
ULTRA\_EXPANDED\_WD (for C) or ULTRAEXP (for COBOL and RPG)

### Style

The style of the font, either roman or italic. The valid values are:

NORMAL\_FS (roman) (for C) or ROMAN (for COBOL and RPG)  
ITALIC\_FS (for C) or ITALIC (for COBOL and RPG)

## Output Parameters

### Font ID

The font ID for use on subsequent Set Font calls.

## Return Codes

See "Font Return Codes" on page 259 for a listing of the return codes for this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

To define a Times New Roman 12 point, bold italic font scaled horizontally to 50 percent of the point size:

```
strcpy(Typeface,"TIMES NEW ROMAN LATIN1");
Codepage="T1V10500"; /*EBCDIC code page 500*/
rc = AFPSetFontType(FONT_OLN);
rc = AFPDefineFontByAttrWithScaling(Codepage, Typeface, 120, 50, BOLD_WT,
                                   MEDIUM_WD, ITALIC_FS, &FontID);
```

- **For the AFP Toolbox COBOL Interface:**

This example selects outline font technology with AFPSetFontType and then defines a font that is 14 point and is scaled horizontally 50 percent.

```
        MOVE FONT-OLN TO AFP-FONT-TYPE
        PERFORM AFPSFTYP.
* DefineFontByAttrWithScaling
        MOVE "T1V10500" TO AFP-CODE-PAGE.
        MOVE "TIMES NEW ROMAN LATIN1" TO AFP-DESCRIPTIVE-NAME.
        MOVE 140 TO AFP-POINT-SIZE.
        MOVE 50 TO AFP-HSCALE
        MOVE MEDIUM TO AFP-WEIGHT.
        MOVE NORML TO AFP-FONT-WIDTH.
        MOVE ROMAN TO AFP-STYLE.
        PERFORM AFPDFNTSC.
        MOVE AFP-FONT-ID TO TIM13TYPE.
```

# Define Font By Name

## Function

Finds a font on your system that matches the specified character set and code page combination and returns a font ID. The font ID is used on subsequent Set Font calls.

For a list of fonts available on your system, contact your system administrator.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPDefineFontByName(  
    char*    in_codePage,  
    char*    in_characterSet,  
    char*    in_codedFont,  
    FontID   out_fontID  
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPDFNTNM.  
    CALL "CBLDefineFontByName"  
        USING  
            BY CONTENT  
                AFP-CODE-PAGE  
                AFP-CHAR-SET  
                AFP-CODED-FONT  
            BY REFERENCE  
                AFP-FONT-ID  
            RETURNING  
                AFP-RET-CODE.  
    MOVE "CBLDefineFontByName" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPDFNTNM      PR          10I 0  EXTPROC('AFPDefineFontByName')  
D  AFPCDEPAG          9  
D  AFPCHRSET          9  
D  AFPCDEFNT          9  
D  AFPFNTID           5I 0
```

## Input Parameters

### Code page

The code page name (the member name in the font library, for example, T1V10500) of the output data. This is not the same as the input code page in which the text data is entered. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, \_ (underscore), #, and @. Code page names must be in uppercase.

The default input code page for all text in AFP Toolbox is assumed to be the multinational ASCII code page T1000850 for workstation systems and EBCDIC T1V10500 for host systems (unless you've changed this value with the Set Input Codepage call).

### Character set

The character set name (member name in the font library). For example, C0H20000 for the Courier 10 point Latin characters. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, \_ (underscore), #, and @. Character set names must be in uppercase.

## Coded font

The coded font name (member name in the font library). This parameter is currently unsupported, it must be set to NULL.

## Output Parameters

### Font ID

The font ID for use on subsequent Set Font calls.

## Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Define a font given a character set and code page combination:

```
CharSet="C0N20000"; /*Courier 10 point normal character set*/
Codepage="T1V10500"; /*EBCDIC code page 500*/
rc = AFPDefineFontByName(Codepage, CharSet, NULL, &FontID);
```

- **For the AFP Toolbox COBOL Interface:**

Define a font given a character set and code page combination:

```
MOVE "T1V10500" TO AFP-CODE-PAGE.
MOVE "C0N20000" TO AFP-CHAR-SET.
PERFORM AFPDFNTNM.
```

- **For the AFP Toolbox RPG Interface:**

Define a font given a character set and code page combination:

```
C*      DTA(8)          CAT(P)      x'00':0      AFPCHRSET
C*      EXSR            AFPDFNTNMR
C*      Z-ADD           AFPFNTID     TIM14NAME
...
**CTDATA DTA
C0N20000
```

8

---

# Define Row

## Function

Creates a row definition for use in a table. Associated Begin Row calls use this row definition. The Begin Field procedure call places fields in a row.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPDefineRow(
    TBHANDLE in_docHandle,
    uint in_minsubrowdepths[],
    long in_topthick,
    long in_botthick,
    uint in_nbrcols,
    uint in_nbrsubrows,
    short far* in_rowarrange,
    uint in_colwidths[],
    short far* out_rowid
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPDROW.
  CALL "CBLDefineRow"
  USING
    BY VALUE
      AFP-DOCUMENT-HANDLE
      AFP-MIN-SUBROW-DEPTH-ARRAY
    BY REFERENCE
      AFP-TOP-THICKNESS
      AFP-BOTTOM-THICKNESS
      AFP-NUMBER-COLUMNS
      AFP-NUMBER-SUBROWS
      AFP-ROW-ARRANGE-ARRAY
      AFP-COLUMN-WIDTH-ARRAY
      AFP-ROW-ID
  RETURNING
    AFP-RET-CODE.
  MOVE "CLBDefineRow" TO AFP-ERRDATA.
  PERFORM CHKSUCC.
```

## Input Parameters

### Document Handle

The handle for the current document returned from the Begin Document call.

### Minimum Subrow Depth Array

The minimum depth of the subrows in a row in the current unit of measure. You can specify a value or use a default that is determined by the field that uses the largest font (that is, the font with the largest default line space). The depth of a subrow includes the top horizontal rule thickness used for the subrow. The depth of the last subrow in a table also includes the bottom horizontal rule thickness. If a subrow shares a common boundary with a table, the table rule is used.

You can specify a specific value or use the default. To use the default, enter one of these values:

#### **DEFAULT for C, or AFP-DEFAULT for COBOL**

Use the default subrow depth.

### Top Thickness

The thickness of the top horizontal rule. If you do not want a rule, specify 0. If a row shares a common boundary with a table, the table rule is used. The rule used for the boundary between two rows is the thicker of the two rules. The value must be greater than or equal to zero.



**Bottom Thickness**

The thickness of the row's horizontal rule. If you do not want a rule, specify 0. If a row shares a common boundary with a table, the table rule is used. The rule used for the boundary between two rows is the thicker of the two rules. The value must be greater than or equal to zero.

**Number of Columns**

The number of vertical divisions within a row. You must specify at least one column.

**Note:** The subrows and columns form an array that is used to describe the contents of the row. The number of elements in the array, computed by multiplying the number of subrows by the number of columns, cannot exceed 64.

**Number of Subrows**

The number of horizontal divisions within a row. Valid values are greater than or equal to zero.

**Note:** The subrows and columns form an array that is used to describe the contents of the row. The number of elements in the array, computed by multiplying the number of subrows by the number of columns, cannot exceed 64.

**Row Arrange Array**

The FIELD-IDs returned by the AFPDFLD calls for the fields that are to be used in each position of the row. Values must be greater than or equal to zero.

Row Arrange Array(n) contains the field IDs of the subrows, starting at the upper left-hand corner and going to the right.

**Column Width Array**

The widths of the columns in the row in the current unit of measure. The width of a column includes the left vertical rule thickness used for the field. The right-most column in a table also includes the right vertical rule thickness in the column width. If a field shares a common boundary with a table, the rule thickness of the table frame is used.

Each column width must be greater than zero, also, the sum of individual column widths should equal the table width specified on the Begin Table call. If data placed in a column with the Put Text in Table Field call exceeds the column width, the call that placed the data generates a non-zero return code.

Column Width Array(n) contains the width of the n<sup>th</sup> column in the row.

**Output Parameters****Row ID**

The field ID for use on associated Begin Row calls.

**Return Codes**

- 3** The document handle is missing or is not valid. Make sure you have called Begin Document before issuing this call.
- 20** An error occurred allocating the table field object. This may be a result of insufficient system memory.
- 23** The rule thickness is not valid. The thickness must be greater than or equal to zero.
- 25** A table field ID in the Row Arrange Array is not valid. Make sure you have called Define Field before issuing this call.
- 27** The number of columns is not valid. There must be at least one column.
- 28** The number of subrows is not valid. Valid values are greater than or equal to zero.
- 34** A Column Width value is incorrect. The column width must be greater than zero.
- 43** No fields have been defined. Call Define Field before issuing this call.

## Define Row

- 57 An error occurred in the row arrangement values in the Row Arrange array. The number of subrows multiplied by the number of columns must equal the number or row arrangement values specified.
- 60 An error occurred in the row arrangement values in the Row Arrange array. Row Arrange values must be greater than or equal to zero.

## Sample Function Call

This command defines characteristics for a row with three columns. The row has these properties:

The font used determines the subrow depth for each subrow.

Use the characteristics for row 1 column 1 that were specified in the Define Field that returned an ID of FIELDH1.

Column 1 is 25 millimeters wide.

Use the characteristics for row 1 column 2 that were specified in the Define Field that returned an ID of FIELDH2.

Column 2 is 70 millimeters wide.

Use the characteristics for row 1 column 3 that were specified in the Define Field that returned an ID of FIELDH3.

Column 3 is 30 millimeters wide

- **For the AFP Toolbox C Library**

```
NbrofSubrows = 1;
NbrofColumns = 3;

TopThick    = MM_2_U1440(.5);
BotThick    = MM_2_U1440(.5);

ColumnWidths[0] = MM_2_U1440(25.0);
ColumnWidths[1] = MM_2_U1440(70.0);
ColumnWidths[2] = MM_2_U1440(30.0);

RowArrange[0] = Hdrfld1; /* MOVE FIELD TO AFP-ROW-ARRANGE */
RowArrange[1] = Hdrfld2;
RowArrange[2] = Hdrfld3;

AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick,
  BotThick, NbrofColumns, NbrofSubrows, &RowArrange[0],
  ColumnWidths, &HdrRowId);
```

- **For the AFP Toolbox COBOL interface:**

This example defines a row with 3 columns and 1 subrow.

```
MOVE MM TO AFP-UNIT-OF-MEASURE.
PERFORM AFPSUNI.
MOVE 3 TO AFP-NUMBER-COLUMNS.
MOVE 1 TO AFP-NUMBER-SUBROWS.
MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH(1).
MOVE FIELDH1 TO AFP-COLUMN-ARRANGE (1, 1).
MOVE 25.0 TO AFP-COLUMN-WIDTH (1).
MOVE FIELDH2 TO AFP-COLUMN-ARRANGE (1, 2).
MOVE 70.0 TO AFP-COLUMN-WIDTH (2).
MOVE FIELDH3 TO AFP-COLUMN-ARRANGE (1, 3).
MOVE 30.0 TO AFP-COLUMN-WIDTH (3).
PERFORM AFPDROW.
```

## Delete Font

### Function

Deletes the specified font. Use this call when your program creates font definitions more than once by looping through multiple Begin Session or Begin Document calls. Font definitions are not automatically freed by the toolbox when the session or document is ended, so if you are looping, you need to free them yourself. Otherwise, you will eventually run out of storage and get unexpected results

Call this procedure once for each font that the application defines. These calls should be executed after End Page for the last page that uses the font. When the last font is deleted, AFP Toolbox releases all other font and codepage storage, and initialize local storage.

**Note:** Only use this call if your program loops through multiple sessions or documents. Normally, fonts are freed when the job ends and you do not need this call.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPDeleteFont(FontID in_fontID);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPDFONT.
    CALL "CBLDeleteFont"
        USING
            BY VALUE
            AFP-FONT-ID.
```

#### For the AFP Toolbox RPG Interface:

```
DAFPDFONT      PR          10I 0 EXTPROC('AFPDeleteFont')
D  AFPFONTID    5I 0 VALUE
```

### Input Parameters

#### Font ID

The ID of the font that is to be deleted.

### Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

### Sample Function Call

Delete the font with identifier MyFontID.

- **For the AFP Toolbox C Library:**

```
AFPDeleteFont(MyFontID);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE TIM10MED TO AFP-FONT-ID.
PERFORM AFPDFONT.
```

- **For the AFP Toolbox RPG Interface:**

```
C          EVAL      AFPFONTID = MyFontID
C          EXSR      AFPDFONTR
```

---

# Delete Page

## Function

Deletes the page whose handle is passed in the first parameter. No AFP output is produced for the given page. The purpose of the Delete Page function is to allow you to format and count pages without actually producing output. For example, you might want to count pages for inserting the phrase "Page *n* of *m*" into the output or sort the output by number of pages.

**Note:** Issue Delete Page **instead** of the End Page call. Once you issue End Page the AFP has already been produced. Do **not** attempt to issue Delete Page after ending the page because the page handle will be invalid.

## Syntax

**For the AFP Toolbox C Library:**

```
int AFPDeletePage(  
    TBHANDLE* inp_pageHandle  
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPDPAGE.  
    CALL "CBLDeletePage"  
        USING  
            BY REFERENCE  
            AFP-PAGE-HANDLE  
            RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLDeletePage" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

## Input Parameters

### Page handle

The handle of the page to be deleted, returned by the Begin Page call.

## Return Codes

- 1      The page handle value is null. You must specify a valid address for this parameter.
- 4      Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 14     This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

```
rc = AFPBgnPage (DocumentHandle, "page1", &PageHandle) ;  
rc = AFPSetFont (PageHandle, TIM10BOLD) ;  
rc = AFPWriteString (PageHandle,AFP_STRING_1,DEFAULT_WS,  
                    DEFAULT_ICS,LEFT_ALIGN,' ',FALSE) ;  
rc = AFPDeletePage(&PageHandle);
```

- **For the AFP Toolbox COBOL Interface:**

```
PERFORM AFPBPAGE.  
    MOVE TIM10BOLD TO AFP-FONT-ID.  
PERFORM AFPSFONT.  
    MOVE "A string o' text" TO AFP-CHARACTER-STRING.  
    MOVE 16 TO AFP-STRING-LENGTH.
```

```
MOVE LEFT-ALIGN TO AFP-ALIGNMENT-OPTION
MOVE FALS TO AFP-POSITION-OPTION.
MOVE -1 TO AFP-EXTRA-VARSPACE.
MOVE -1 TO AFP-EXTRA-ICSPACE.
MOVE "." TO AFP-ALIGNMENT-CHAR.
PERFORM AFPWRITE.
PERFORM AFDPAGE.
```

---

## End Bar Code

### Function

Ends a bar code previously started with Begin Bar Code. At this time the bar code is placed in the page.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPEndBarCode(  
    TBHANDLE      in_pageHandle,  
    TBHANDLE*     inp_bcHandle  
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPEBCD.  
    CALL "CBLEndBarCode"  
        USING  
            BY VALUE  
                AFP-PAGE-HANDLE  
            BY REFERENCE  
                AFP-BARCODE-HANDLE  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLEndBarCode" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPEBCD          PR          10I 0  EXTPROC('AFPEndBarCode')  
D  AFPPAGHDL      *          *  VALUE  
D  AFPBCHDL       *          *
```

### Input Parameters

#### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

#### Bar code handle

The handle for the bar code to place, returned by the Begin Bar Code call. This parameter is set to null by the End Bar Code call.

### Return Codes

- 4      Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 17     Bar code handle is not pointing to a valid bar code object. Make sure you have called Begin Bar Code before issuing this call.

### Sample Function Call

See “Begin Bar Code” on page 66 for a complete Bar Code example.

## End Box

### Function

Ends a box at the current location. The box must have been previously started with a Begin Box request.

The depth of the box is calculated from the top of the topmost rule to the top of the bottommost rule. In other words, the box depth includes the width of the top rule but not the bottom rule.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPEndBox(
    TBHANDLE      in_pageHandle,
    TBHANDLE*     inp_boxHandle
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPEBOX.
    CALL "CBLEndBox"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
            BY REFERENCE
                AFP-BOX-HANDLE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLEndBox" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPEBOX      PR          10I 0 EXTPROC('AFPEndBox')
D AFPPAGHDL          *    VALUE
D AFPBOXHDL          *
```

### Input Parameters

#### Page handle

The handle for the page. Page handle is returned by the Begin Page call.

#### Box handle

The handle for this box. Box handle is returned by the Begin Box call when you started the box. When returned, this parameter is set to null.

### Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 5 The **inp\_boxHandle** parameter is null or does not point to a valid box. Make sure you have started the box with Begin Box before trying to end it.

### Sample Function Call

See “Begin Box” on page 73 for a complete Box example.

## End Color

### Function

Ends a rectangular block of color previously started with Begin Color.

**Note:** Use of this function requires microcode support in your printer.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPEndColor(  
    TBHANDLE    in_pageHandle,  
    TBHANDLE*   inp_colorHandle  
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPECLR.  
    CALL "CBLEndColor"  
    USING  
        BY VALUE  
            AFP-PAGE-HANDLE  
        BY REFERENCE  
            AFP-COLOR-HANDLE  
    RETURNING  
        AFP-RET-CODE.  
    MOVE "CBLEndColor" TO AFP-ERRDATA.  
  
    PERFORM CHKSUCC.
```

### Input Parameters

#### Page Handle

The handle for the page to contain this color block. Page handle is returned by the Begin Page call.

#### Color Handle

The handle for the color block returned from Begin Color.

### Return Codes

- 4      The page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 5      The color handle is null or empty.

### Sample Function Call

See “Begin Color” on page 75 for a complete color example.



## End Document

### Function

Ends the document and causes the remainder of the data stream to be written to the output destination.

**Note:** You must issue End Page for all pages before issuing End Document or the pages are not placed in the output. If you requested buffered output on the Begin Document call, then you need to follow End Document with repeated calls to Get Buffer until `outp_moreData` is FALSE. See “Get Buffer” on page 150 for an example.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPEndDoc(
    TBHANDLE*    inp_docHandle
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPEDOC.
    CALL "CBLEndDoc"
        USING
            BY REFERENCE
            AFP-DOCUMENT-HANDLE
            RETURNING
            AFP-RET-CODE.
    MOVE "CBLEndDoc" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPEDOC          PR          10I 0 EXTPROC('AFPEndDoc')
D  AFPDOCHDL          *
```

### Input Parameters

#### Document handle

The handle of the document to end. Document handle is returned by the Begin Document call. When returned, this parameter is set to null.

### Return Codes

- 3** The document handle is missing or is not valid. Make sure you have called Begin Document before issuing this call.

### Sample Function Call

See “Begin Document” on page 78 for a complete Document example.

## End Field

### Function

Ends a field of data in a table

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPEndField(  
    TBHANDLE in_tb1Handle  
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPEFLD.  
    CALL "CBLEndField"  
    USING  
    BY VALUE  
    AFP-TABLE-HANDLE  
    RETURNING  
    AFP-RET-CODE.  
MOVE "CBLEndField" TO AFP-ERRDATA.  
PERFORM CHKSUCC.
```

### Input Parameters

#### Table Handle

The handle for the associated table, returned from the Begin Table call.

### Return Codes

- 1 The table object handle is invalid or null. Call Begin Table before issuing End Table.

### Sample Function Call

For a complete example, see “Begin Field” on page 82.

- **For the AFP Toolbox C Library:**

```
AFPEndField(hTb1);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPEFLD.
```

## End Group

### Function

Ends a logical group of pages previously started with Begin Group. If no group is active this request is ignored.

Groups of pages cannot be nested or overlapped. Each group must be ended before another can begin.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPEndGroup(
    TBHANDLE    in_docHandle
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPEGRP.
    CALL "CBLEndGroup"
        USING
            BY VALUE
            AFP-DOCUMENT-HANDLE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLEndGroup" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

#### For the AFP Toolbox RPG Interface:

```
DAFPEGRP      PR          10I 0 EXTPROC('AFPEndGroup')
D  AFPDOCHDL          *    VALUE
```

### Input Parameters

#### Document handle

The handle for the document. Document handle is returned by the Begin Document call.

### Return Codes

- 3**      The document handle is missing or is not valid. Make sure you have called Begin Document before issuing this call.

### Sample Function Call

See “Begin Group” on page 84 for a complete Group example.

## End Image

---

## End Image

### Function

Ends the image object previously started with the corresponding Begin Image call.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPEndImage(  
    TBHANDLE      in_pageHandle,  
    TBHANDLE*     inp_imageHandle  
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPEIMG.  
    CALL "CBLEndImage"  
        USING  
            BY VALUE  
                AFP-PAGE-HANDLE  
            BY REFERENCE  
                AFP-IMAGE-HANDLE  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLEndImage" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

### Input Parameters

#### Page handle

The handle for the page, returned by the Begin Page call.

#### Image handle

The handle for the image, returned from the Begin Image call. This parameter is set to null by End Image

### Return Codes

- 4      Page handle is not pointing to a valid page. make sure you have called Begin Page before issuing this call.

### Sample Function Call

See “Image Data” on page 158 for a complete image example.

## End Page

### Function

Ends the page. If output is being buffered (returned to the program), then you need to follow End Page with repeated Get Buffer calls until “outp\_moreData” is false. If not buffered, the page is output to the appropriate file (or spool) after this call has been received. No more data can be placed in the page once it is ended. See the example in “Get Buffer” on page 150 for more information.

Do not issue this call within a paragraph or a table.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPEndPage(
    TBHANDLE*    inp_pageHandle
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPEPAGE.
    CALL "CBLEndPage"
        USING
            BY REFERENCE
            AFP-PAGE-HANDLE
            RETURNING
            AFP-RET-CODE.
    MOVE "CBLEndPage" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPEPAG      PR          10I 0 EXTPROC('AFPEndPage')
D  AFPPAGHDL          *
```

### Input Parameters

#### Page handle

The handle of the page to terminate. Page handle is returned by the Begin Page call. This parameter is set to null by End Page unless output is being buffered. In that case the handle can be used on Get Buffer calls to retrieve the structured fields for the page contents.

### Return Codes

- 1** The page handle value is null. You must specify a valid address for this parameter.
- 4** Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 14** This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.

### Sample Function Call

See “Begin Page” on page 91 for a complete Page example.

## End Paragraph

---

## End Paragraph

### Function

Ends the previous paragraph started with the Begin Paragraph call.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPEndPgraph(  
    TBHANDLE    in_pageHandle  
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPEPARA.  
    CALL "CBLEndPgraph"  
        USING  
        BY VALUE  
        AFP-PAGE-HANDLE  
        RETURNING  
        AFP-RET-CODE.  
    MOVE "CBLEndPgraph" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPEPARA      PR          10I 0 EXTPROC('AFPEndPgraph')  
D AFPPAGHDL      *      VALUE
```

### Input Parameters

#### Page handle

The handle for this page. Page handle is returned by the Begin Page call.

### Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

### Sample Function Call

See “Begin Paragraph” on page 93 for an example of paragraph code.

## End Row

### Function

Ends a row in a table.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPEndRow(
    TBHANDLE in_tblHandle,
    uint in_rowid,
    unsigned short far* out_currtbldepth
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPEROW.
    CALL "CBLEndRow"
    USING
        BY VALUE
        AFP-TABLE-HANDLE
    AFP-ROW-ID
        BY REFERENCE
        AFP-CURRENT-TABLE-DEPTH
    RETURNING
        AFP-RET-CODE.
    MOVE "CBLEndRow" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### Table Handle

The handle for the associated table, returned from the Begin Table call.

#### Row ID

The handle for the row to end, returned by the Begin Row call.

### Output Parameters

#### Current Table Depth

The depth of the table in the current unit of measure.

### Return Codes

- 1**      The table object handle is invalid or null. Call Begin Table before issuing this call.
- 25**     No valid field as been allocated. Allocate at least one field object by calling Define Field.
- 26**     The row ID parameter is invalid. Make sure you have called Begin Row before issuing this call.
- 27**     Too many columns have been specified. You cannot specify more than 64 columns.
- 34**     No valid column width object was found for the row. The column width object is allocated for the row after a successful call to Begin Row. Ensure this call ended with no errors.
- 37**     A table depth error occurred for the current page. End this row on a different page.
- 66**     The current field is not ended. Issue End Field before calling End Row.

### Sample Function Call

This command ends the current row.

- **For the AFP Toolbox C Library:**

## End Row

```
AFPEndRow(hTbl, HdrRowId, &CurrTableDepth);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPEROW.
```



## End Rule

### Function

Ends previous rule with the corresponding Begin Rule at the current position.

**Note:** Use Begin and End Rule functions when you want to draw a variable length rule. If you know the rule length, use Put Horizontal Rule or Put Vertical Rule instead for efficiency.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPEndRule(
    TBHANDLE    in_pageHandle,
    TBHANDLE*    inp_ruleHandle
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPERULE.
    CALL "CBLEndRule"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
            BY REFERENCE
                AFP-RULE-HANDLE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLEndRule" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPERULE      PR          10I 0  EXTPROC('AFPEndRule')
D AFPPAGHDL    *          *  VALUE
D AFPRULHDL    *          *
```

## Input Parameters

### Page handle

The handle for this page. Page handle is returned by the Begin Page call.

### Rule handle

The handle for the rule. Rule handle is returned by the Begin Rule call. This parameter is set to null by End Rule.

## Return Codes

- 4** Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 6** Rule handle is not pointing to a rule object. Make sure you have called Begin Rule before issuing this call.

## Sample Function Call

See “Begin Rule” on page 97 for an example of drawing a rule.

# End Session

## Function

Normal termination of the AFP Toolbox session. You cannot issue any other AFP Toolbox procedure call after End Session has been issued except for another Begin Session.

## Syntax

**For the AFP Toolbox C Library:**

```
int AFPEndSession(  
    TBHANDLE*    inp_sessHandle  
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPEND.  
    CALL "CBLEndSession"  
        USING  
        BY REFERENCE  
        AFPAPI-HANDLE  
        RETURNING  
        AFP-RET-CODE.  
    MOVE "CBLEndSession" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPEND          PR          10I 0 EXTPROC('AFPEndSession')  
D  AFPAPIHDL          *
```

## Input Parameters

### Session handle

The handle for this AFP Toolbox session. Session handle is returned by the Begin Session call. This parameter is set to null by End Session.

## Return Codes

- 2 Session handle is null or is not valid. Make sure you have issued Begin Session before you try to end it.

## Sample Function Call

See “Begin Session” on page 100 for a complete Session example.

## End Shade

### Function

Ends a shaded rectangle at the current location. The shaded area must have been previously started with a Begin Shade request.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPEndShade(
    TBHANDLE      in_pageHandle,
    TBHANDLE*     inp_shadeHandle
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPESH.D.
    CALL "CBLEndShade"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
            BY REFERENCE
                AFP-SHADE-HANDLE
    MOVE "CBLEndShade" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### Page handle

The handle for this page. Page handle is returned by the Begin Page call.

#### Shade handle

The handle for this shaded area. Shade handle is returned by the Begin Shade call when you started the shaded area. When returned, this parameter is set to null by End Shade.

### Return Codes

- 4      Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 5      The **inp\_shadeHandle** value is null or does not point to a shaded area. Make sure you have started the shaded area with Begin Shade before trying to end it.

### Sample Function Call

See “Begin Shade” on page 101 for a complete shading example.

## End Table

### Function

Ends a table and returns the depth of the table, including the depth of the horizontal rule, in the current units of measure. The current position at the end of a table is at the bottom-left corner of the table.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPEndTable(  
    TBHANDLE* in_tblHandle,  
    unsigned short far* out_tbldepth  
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPETBL.  
    CALL "CBLEndTable"  
    USING  
    BY VALUE  
        AFP-TABLE-HANDLE  
    BY REFERENCE  
        AFP-CURRENT-TABLE-DEPTH  
    RETURNING  
        AFP-RET-CODE.  
    MOVE "CBLEndTable" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

### Input Parameters

#### Table Handle

The handle of the table ended, returned from the Begin Table call. When returned, this parameter is set to 0.

### Output Parameters

#### Table Depth

The depth of the table in the current unit of measure.

### Return Codes

- 1        The table object handle is invalid or null. Call Begin Table before issuing this call.
- 27      Too many columns have been specified. You cannot specify more than 64 columns.

### Sample Function Call

This command ends a table.

- **For the AFP Toolbox C Library:**

```
AFPEndTable(&hTbl, &CurrTableDepth);
```

- **For the AFP Toolbox COBOL interface:**

```
PERFORM AFPETBL.
```

## End Underscore

### Function

The End Underscore function ends underscoring of text previously started with a Begin Underscore request.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPEndUScore(TBHANDLE in_pageHandle);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPEUSC.  
  CALL "CBLEndUScore"  
  USING BY VALUE  
  AFP-PAGE-HANDLE  
  RETURNING  
  AFP-RET-CODE.
```

### Input Parameters

#### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Sample Function Call

For a complete example, see “Begin Underscore” on page 106.

---

## Get Buffer

### Function

Returns the next buffer of output for the page. This call is only valid if output is being returned to the program instead of to an output file. Before issuing Get Buffer, a Begin Document call specifying TO\_OUTPUT\_BUFFER must be issued to request that output goes to a buffer and what the maximum buffer length is. One AFP structured field is returned for each Get Buffer call.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPGetBuffer(
    TBHANDLE    in_currentHandle,
    int*        outp_bufferLength,
    boolean*    outp_moreData
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPGBUF.
    CALL "CBLGetBuffer"
        USING
            BY VALUE
                AFP-CURRENT-HANDLE
            BY REFERENCE
                AFP-BUFFER-LENGTH
                AFP-MORE-RECORDS
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLGetBuffer" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

#### For the AFP Toolbox RPG Interface:

```
DAFPGBUFF      PR          10I 0  EXTPROC('AFPGetBuffer')
D AFPCURHDL          *      VALUE
D AFPBUFLN          10I 0
D AFPMORERC          10I 0
```

### Input Parameters

#### Current handle

The handle for the current document or page returned from the Begin Document or the Begin Page call. If Get Buffer is specified after End Page, this should be the handle of the last ended page. If Get Buffer is specified after End Document, use the document handle.

### Output Parameters

#### Buffer length

Number of bytes returned in the buffer.

#### More data

Indicates whether there is more data remaining in the document. You should issue GetBuffer calls repeatedly until outp\_moreData is FALSE. Do not process the buffer if outp\_moreData is FALSE.

### Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Retrieve all the records for this page:

```
#ifndef MVS
static char outfile[] = "e:\\tbxout\\filetst.afp";
static char myoutput[] = "e:\\tbxout\\bufftst.afp";
#else
static char outfile[] = "BUFTSTDD";
static char myoutput[] = "bufftst.afp";
#endif
byte mybuff[8*1024]; /* buffer to contain AFP records */ FILE* out;
/* if you want buffered output, change the next line to TO_OUTPUT_BUFFER*/
OUTTYPE output = TO_OUTPUT_FILE;
if (output == TO_OUTPUT_BUFFER)
{
#ifndef MVS
out = fopen(myoutput, "wb");
#else
/* file should be allocated with lrecl 8205 */
out = fopen(myoutput, "wb, recfm=v, type=record");
#endif
rc = AFPBgnDoc (SessionHandle,"SAMPLE","MY TEST SAMPLE",
               output,outfile,mybuff,&DocumentHandle) ;
rc = AFPBgnPage (DocumentHandle,&pagecount,&PageHandle) ;
...
rc = AFPEndPage (&PageHandle) ;

/* after each page is ended the AFP output must be retrieved */
/* the 'morestuff' flag will be set to false by Toolbox when */
/* there are no more output records to process. */
if (output == TO_OUTPUT_BUFFER)
{
    morestuff = TRUE;
    do
    {
        rc = AFPGetBuffer(&PageHandle,&bufflen,&morestuff);
        if (morestuff)
            count = fwrite(mybuff,1,bufflen,out); /* write to output file */
    } // end do while
    while(morestuff);
} // end if buffered output
rc = AFPEndDoc (&DocumentHandle) ;
if (output == TO_OUTPUT_BUFFER) /* get EDT and any other SFs */
{
    morestuff = TRUE;
    do
    {
        rc = AFPGetBuffer(&DocumentHandle,&bufflen,&morestuff);
        if (morestuff)
            count = fwrite(mybuff,1,bufflen,out); /* write to output file */
    } // end do while
    while(morestuff);
} // end if buffered output
```

- **For the AFP Toolbox COBOL Interface:**

For an example of buffered output, see “CBLGetBuffer Program” on page 382.

---

# Graphic Inline

## Function

Reads a previously created GOCA object and includes it inline in the data stream at the current position. The graphic filename must be fully qualified and be accessible with appropriate permission for this application to read the file. The graphic can reside in a page segment or it can be a standalone object.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPGraphicInline(
    TBHANDLE    in_pageHandle,
    char*        in_graphicFile
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPGINL.
  CALL      "CBLGraphicInline"
  USING
  BY VALUE
    AFP-PAGE-HANDLE
  BY CONTENT
    AFP-GRAPHIC-FILE
  RETURNING
    AFP-RET-CODE.
  MOVE "CBLGraphicInline" TO AFP-ERRDATA.
  MOVE SPACES TO AFP-GRAPHIC-FILE(AFP-NAME-LENGTH:1)
  PERFORM CHKSUCC.
```

## Input Parameters

### Page handle

The handle for the page that was processed. Page handle is returned by the Begin Page call.

### Graphic file

The name of the file containing the graphic object. For MVS, this is the DD name of the file; for all other environments it is the actual file name.

## Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Include a graphic named "goctest.psg" from the fld\resource directory on the c: drive.

```
rc = AFPGraphicInline(PageHandle,"c:\\fld\\resource\\goctest.psg");
```

- **For the AFP Toolbox COBOL Interface:**

```
*-----
* Do a set position to 1 inch down, 1 inch across
* and place a GOCA object
*-----
* SetPos
      MOVE 1440 TO AFP-X-COORDINATE.
      MOVE 1440 TO AFP-Y-COORDINATE.
      MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
      MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
```



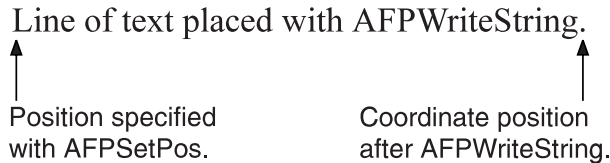
```
PERFORM AFPSPOS.  
* GraphicInline takes one parm, in MVS this is DDNAME  
  MOVE "MYGOCA"      TO AFP-GRAPHIC-FILE.  
PERFORM AFPGINL.
```

---

# Horizontal Move

## Function

Move inline horizontally relative to the current inline coordinate position. Positive values move the position to the right; negative values move the position to the left. This is not necessarily the last position specified with the SetPos command; it is a move relative to the last presentation position in the output. For example, if the Set Position followed by Write String<sup>4</sup> commands have been issued, the last position you set is at the *beginning* of the text that was written, but the coordinate position is at the *end* of the text. The following figure illustrates the difference:



The new current position is at the specified position (inline), and it is unchanged in the baseline direction.

**Note:** Do not use this call within a paragraph or table.

## Syntax

**For the AFP Toolbox C Library:**

```
int AFPHMove(
    TBHANDLE    in_page_Handle,
    short       in_incrementValue
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPHMOVE.
    CALL "CBLHMove"
        USING
        BY VALUE
        AFP-PAGE-HANDLE
        AFP-X-COORDINATE
        RETURNING
        AFP-RET-CODE.
    MOVE "CBLHmove" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPHMOVE      PR          10I 0 EXTPROC('AFPHMove')
D AFPPAGHDL          *      VALUE
D AFPXCRD           10I 0 VALUE
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Increment value

The amount of the horizontal move. The value can be positive (move right) or negative (move left).

---

4. This is only true if you specified False for the "Same Position" parameter on the Write String call.

## Return Codes

- 4        Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 14      This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Add two inches to the current horizontal position:

```
HDist = IN_2_U1440(2);
rc = AFPHMove(PageHandle, HDist);
```

- **For the AFP Toolbox COBOL Interface:**

Add two inches to the current horizontal position:

```
MOVE 2880 TO AFP-X-COORDINATE.
PERFORM AFPHMOVE.
```

- **For the AFP Toolbox RPG Interface:**

Add two inches to the current horizontal position:

C	Z-ADD	2880	AFPXCRD
C	EXSR	AFPHMOVER	

# Horizontal Move To

## Function

Move inline horizontally to a specified position. The new current position is at the specified position inline, and unchanged in the baseline (Y) direction.

**Note:** Do not issue this call within a paragraph or table.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPHMoveTo(
    TBHANDLE    in_pageHandle,
    ushort      in_inlinePosition
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPHMOVETO.
    CALL "CBLHMoveTo"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-X-COORDINATE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLHMoveTo" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPHMOVTO      PR          10I 0 EXTPROC('AFPHMoveTo')
D AFPPAGHDL      *          *  VALUE
D AFPXCRD        10I 0  VALUE
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Inline position

The new horizontal position.

## Return Codes

- 4      Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 14     This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Set the current horizontal position at two inches:

```
HDist = IN_2_U1440(2);
rc = AFPHMoveTo(PageHandle, HDist);
```

- **For the AFP Toolbox COBOL Interface:**

Set the current horizontal position at two inches:

MOVE 2880 TO AFP-X-COORDINATE.  
PERFORM AFPHMOVETO.

- **For the AFP Toolbox RPG Interface:**

Set the current horizontal position at two inches:

C	Z-ADD	2880	AFPXCRD
C	EXSR	AFPHMOVTOR	

---

# Image Data

## Function

Include image data records into an image that was started with Begin Image.

## Syntax

**For the AFP Toolbox C Library:**

```
int AFPIImageData(  
    TBHANDLE      in_pageHandle,  
    TBHANDLE      in_imageHandle,  
    byte*         in_imageBuffer,  
    int           in_imageBufferLen,  
    CompressionType in_compr,  
    ushort        in_recordingAlg,  
    ushort        in_bitOrder  
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPIDATA.  
    CALL "CBLImageData"  
        USING  
            BY VALUE  
                AFP-PAGE-HANDLE  
                AFP-IMAGE-HANDLE  
            BY CONTENT  
                AFP-IMAGE-BUFFER  
            BY VALUE  
                AFP-BUFFER-LENGTH  
                AFP-COMPRESSION-TYPE  
                AFP-RECORDING-ALG  
                AFP-BIT-ORDER  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLImageData" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPIMGDAT      PR          10I 0  EXTPROC('AFPIImageData')  
D AFPPAGHDL      *          *  VALUE  
D AFPMGHDL      *          *  VALUE  
D AFPMBUFF      8205  
D AFPMBLEN      10I 0  VALUE  
D AFPMCOMP      10I 0  VALUE  
D AFPMRECD      10I 0  VALUE  
D AFPMBTOR      10I 0  VALUE
```

## Input Parameters

### Page handle

The handle for the page. Page handle is returned by the Begin Page call.

### Image handle

The handle for the image. Image handle is returned by the Begin Image call.

### Image buffer

Raw image data (data without AFP structured fields) to include in the image object. Only simple image data can be specified. Banded image data, numbered image data, and tiled image data are not supported.

**Image buffer length**

Number of bytes in the image data buffer.

**Compression type**

Specifies which algorithm to use for compressing the image data. Compression type does not compress the data. The valid values are:

```
MMR    = 0x01,      /* IBM MMR (Modified CCITT READ Algorithm */
RAW     = 0x03,      /* No compression */
RL4     = 0x06,      /* Run-Length 4 */
ABIC    = 0x08,      /* Bi-level Q-coder */
CABIC   = 0x0A,      /* Concatenated ABIC */
CCOMP   = 0x0B,      /* Color compression used by OS/2 */
G3MH    = 0x80,      /* CCITT T.4 G3 (1D fax coding) */
G3MR    = 0x81,      /* CCITT T.4 G3 (2D fax coding) */
G4MMR   = 0x82,      /* CCITT T.6 G4 (2D fax coding) */
JPEG    = 0x83      /* ISO/CCITT JPEG algorithms */
```

**Recording algorithm**

Specifies how the image data is recorded. Valid values are:

```
1 RIDIC (Recording Image Data Inline Coding)
3 Bottom to top
```

**Note:** You must specify the actual value used to record the image data.

**Bit order**

Specifies the bit order within each image data byte. Zero indicates left-to-right order one indicates right-to-left order. **Bit order** is ignored because it only applies to IOCA Function Set 11 (FS11). The Toolbox currently only supports Function Set 10 (FS10) because not all IPDS™ printers support FS11.

**Return Codes**

- 1 The **inp\_imageHandle** value is null. You must specify a valid address for this parameter.
- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

**Sample Function Call**

See “Begin Image” on page 86 for a complete image example.

# Image Inline

## Function

Reads a previously created IOCA or IM1 image and includes it inline in the data stream at the current position. The image filename must be fully qualified and be accessible with appropriate permission for this application to read the file. The image can reside in a page segment, but it must be the only image in the file or else all images in the page segment are included at the same position on the page interfering with each other.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPIImageInline(  
    TBHANDLE    in_pageHandle,  
    char*       in_imageFile  
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPIINL.  
    CALL      "CBLImageInline"  
          USING  
            BY VALUE  
              AFP-PAGE-HANDLE  
            BY CONTENT  
              AFP-IMAGE-FILE  
          RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLImageInline" TO AFP-ERRDATA.  
    MOVE SPACES TO AFP-IMAGE-FILE(AFP-NAME-LENGTH:1)  
    PERFORM CHKSUCC.
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Image file

The name of the file containing the image. For MVS, this is the DD name of the file; for all other environments it is the actual file name.

## Return Codes

- 4      Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

### For the AFP Toolbox C Library:

This example includes an external file containing an IOCA page segment.

```
rc = AFPIImageInline(PageHandle, "c:\\fld\\resource\\iocatst.psg");
```

### For the AFP Toolbox COBOL Interface:

This example moves the current position to one inch, four inches and includes an IOCA object.



```
*-----  
* Do a set position to 4 inch down, 1 inch across  
* and place an image  
*-----  
* SetPos  
    MOVE 1440  TO AFP-X-COORDINATE.  
    MOVE 6480  TO AFP-Y-COORDINATE.  
    MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.  
    MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.  
    PERFORM AFPSPOS.  
* ImageInline takes one parm, the filename  
* In MVS this is the DDNAME  
    MOVE "MYIOCA" TO AFP-IMAGE-FILE.  
    PERFORM AFPIINL.
```

## Include Object

---

## Include Object

### Function

Includes a reference to an image, graphic, bar code, or non-AFP object at the current position and specifies the size, rotation, mapping option, and offset of the printed object.

**Note:** The Include Object function requires appropriate PSF support to print these objects.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPIncObj(  
    TBHANDLE      in_pageHandle,  
    char*         in_objectName,  
    ObjectType     in_objectType,  
    long           in_objectWidth,  
    long           in_objectDepth,  
    DegRot         in_objectRotation,  
    MappingOption  in_mappingOption,  
    long           in_xOffset,  
    long           in_yOffset,  
    OtherObjectType in_otherObjType  
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPIOBJ.  
    CALL "CBLIncObj"  
        USING  
            BY VALUE  
                AFP-PAGE-HANDLE  
            BY CONTENT  
                AFP-OBJECT-NAME  
            BY VALUE  
                AFP-OBJECT-TYPE  
                AFP-AREA-XSIZE  
                AFP-AREA-YSIZE  
                AFP-OBJECT-ROTATION  
                AFP-OBJECT-MAPPING-OPTION  
                AFP-AREA-XPOS  
                AFP-AREA-YPOS  
                AFP-OTHER-OBJECT-TYPE  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLIncObj" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

#### For the AFP Toolbox RPG Interface:

```
DAFPINCOBJ      PR          10I 0  EXTPROC('AFPIncObj')  
D AFPPAGHDL      *          *    VALUE  
D AFPOBNAM       9  
D AFPOBTYP      10I 0  
D AFPOBXSIZ     10I 0  
D AFPOBYSIZ     10I 0  
D AFPOBRTAT     10I 0  
D AFPOBMOPT     10I 0  
D AFPOBXPOS     10I 0  
D AFPOBYPOS     10I 0
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Object name

The name of the object member in the object library.

### Object type

The type of the object. Valid values are:

#### **PAGE\_SEGMENT\_OT (for C) or PAGE-SEGMENT-OT (for COBOL)**

A MO:DCA page segment resource object containing an IOCA image. Page segment objects containing IM1 image cannot take advantage of the capabilities of Include Object and should be referenced with Include Page Segment instead.

**Note:** Only IOCA page segments can be trimmed or scaled. Trimming and scaling page segments that contain IM1 image data is not supported.

#### **GRAPHICS\_OT (for C), GRAPHICS-OT for COBOL, or GRAPHICOT (for RPG)**

A graphics object with MO:DCA syntax as defined in *Graphics Object Content Architecture Reference*

#### **IMAGE\_OT (for C), IMAGE-OT (for COBOL), or IMAGEOT (for RPG)**

An image object with MO:DCA syntax as defined in *Image Object Content Architecture Reference*

#### **BAR\_CODE\_OT (for C), BAR-CODE-OT (for COBOL), or BARCODEOT (for RPG)**

A bar code object with MO:DCA syntax as defined in *Bar Code Object Content Architecture Reference*

#### **OTHER\_OBJECT\_DATA\_OT (for C and C++), OTHER-OBJECT-DATA-OT (for COBOL), or OTHEROT (for RPG)**

Object data to include that is not defined by an IBM presentation architecture. Most “other” types of objects cannot take advantage of the mapping option functions of Include Object.

### Other object type

Use this parameter to fully identify the object type when Object Type is set to Other. Values are:

Value	Description
EPS_00T (for C), EPS-00T (for COBOL)	EPS
TIFF_00T (for C), TIFF-00T (for COBOL)	TIFF
DIB_WIN_00T (for C), DIB-WIN-00T (for COBOL)	DIB, Windows version
DIB_OS2_00T (for C), DIB-OS2-00T (for COBOL)	DIB, OS/2 PM version
PCX_00T (for C), PCX-00T (for COBOL)	PCX
GIF_00T (for C), GIF-00T (for COBOL)	GIF
JFIF_00T (for C), JFIF-00T (for COBOL)	JFIF
PDF_SP0_00T (for C), PDF-SP0-00T (for COBOL)	PDF Single Page Object
PCL_PO_00T (for C), PCL-P0-00T (for COBOL)	PCL Page Object
EPS_TRAN_00T (for C), EPS-TRAN-00T (for COBOL)	EPS with Transparency
PDF_TRAN_00T (for C), PDF-TRAN-00T (for COBOL)	PDF with Transparency

### Object area width

The width of the object area in which to map the object data. This parameter is used to tell the presentation system the width used when placing the object on the page (for example, if the object is to be scaled).

## Include Object

### Object area depth

The depth of the object area in which to map the object data. This parameter is used to tell the presentation system the depth used when placing the object on the page (for example, if the object is to be scaled).

### Object area rotation

The rotation of the object clockwise around the object's origin. The valid values are:

**DEG\_0 (for C), DEG-0 (for COBOL), or DEG0 (for RPG)**

The object area is not rotated.

**DEG\_90 (for C), DEG-90 (for COBOL), or DEG90 (for RPG)**

The object area is rotated 90 degrees clockwise.

**DEG\_180 (for C), DEG-180 (for COBOL), or DEG180 (for RPG)**

The object area is rotated 180 degrees clockwise.

**DEG\_270 (for C), DEG-270 (for COBOL), or DEG270 (for RPG)**

The object area is rotated 270 degrees clockwise.

### Object mapping options

The mapping of the object data within its area.

For OTHER type objects, the mapping options are ignored. For MO:DCA objects and page segments, the valid values are:

**POSITION\_MOPT (for C), POSTION-DEFAULT (for COBOL), or POSDEFAULT (for RPG)**

Position the upper left corner of the object's presentation space with the object's content origin.

**POSITION\_AND\_TRIM\_MOPT (for C), POSTION-AND-TRIM (for COBOL), or POSTRIM (for RPG)**

Position the object at the location specified with Object Area Position within the dimensions specified with Object Area Size and trim what falls outside the area.

**SCALE\_TO\_FIT\_MOPT (for C), SCALE-TO-FIT (for COBOL), or SCALE2FIT (for RPG)**

Center the object within the area dimension specified with Object Area Size and scale the object to fit within the area.

**CENTER\_AND\_TRIM\_MOPT (for C), CENTER-AND-TRIM (for COBOL), or CENTRTRIM (for RPG)**

Center the object within the area dimension specified with Object Area Size and trim what falls outside the area.

**IMAGE\_POINT\_TO\_PEL\_MOPT (for C)**

Position the object at the object area origin within the dimensions specified in **object area width** and in **object area depth**, and trim what falls outside the area. No resolution correction is done, that is, each image point is mapped to a pel.

**MIGRATION-MAP-1 (for COBOL) or POINT2PEL (for RPG)**

This mapping is for IM image migration. The origin of the IO image presentation space is positioned at the origin of the object area. Each image point in the presentation space is mapped to a presentation device pel. Any portion of the image that falls outside the object area is trimmed.

**IMAGE\_DOUBLE\_DOT\_MOPT (for C)**

Position the object at the object area origin within the dimensions specified in **object area width** and in **object area depth**, and trim what falls outside the area. Each image point is doubled. No resolution correction is done, that is, each image point is mapped to a pel.

**MIGRATION-MAP-2 (for C) or DOUBLEDOT (for COBOL)**

This mapping is for IM image migration. The origin of the IO image presentation space is positioned at the origin of the object area. Each image point in the presentation space is

doubled in both directions, resulting in four new image points. The four new image points are then mapped to presentation device pels. Any portion of the image that falls outside the object area is trimmed.

#### **REPLICATE\_AND\_TRIM\_MOPT (for C), REPLCT-AND-TRIM (for COBOL), or REPLCTRM (for RPG)**

Position the data object presentation space in the object area so that its origin is coincident with the origin of the object area and its size is unchanged. The data object presentation space or graphics presentation space window in the case of GOCA is then replicated in the X and Y directions of the object area until the object area is filled.

#### **Object offset (Object x offset and object y offset)**

The position of the object data relative to the object area origin. It is only valid for an object mapping option of position and trim. For all other mapping options, this parameter is ignored. You can specify a specific value or use the default position in the object.

#### **Object area position**

Specifies the x and y positions of the object origin relative to the page origin.

## **Return Codes**

- 4** Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 23** The object type is not valid. See the Object type parameter for valid values.
- 24** Object type is other, but the other object type is not valid. See the Other object type parameter for valid values.

## **Sample Function Call**

#### **• For the AFP Toolbox C Library:**

Include an IOCA page segment object at the current position. Object area width is 2.83 inches, object area depth is 1.83 inches. Mapping option is scale-to-fit. Object x and y offsets inside in the object are used.

```
rc = AFPInc1Obj(PageHandle,"S1IOCAMU",PAGE_SEGMENT_OT,
4080,2640,DEG_0,
SCALE_TO_FIT_MOPT, 0xFFFFFFFF,0xFFFFFFFF);
OMS_checkReturnCode (rc, "AFPInc1Obj");
```

#### **• For the AFP Toolbox COBOL Interface:**

This example creates a reference to an object named MYOBJ, positioning it at the current position.

```
MOVE "MYOBJ" TO AFP-OBJECT-NAME.
MOVE BAR-CODE-OT TO AFP-OBJECT-TYPE.
MOVE 4320 TO AFP-AREA-XSIZE.
MOVE 4320 TO AFP-AREA-YSIZE.
MOVE DEG-0 TO AFP-OBJECT-ROTATION.
MOVE POSITION-AND-TRIM
TO AFP-OBJECT-MAPPING-OPTION.
MOVE 0 TO AFP-AREA-XPOS.
MOVE 0 TO AFP-AREA-YPOS.
```

PERFORM AFPIOBJ.

#### **• For the AFP Toolbox RPG Interface:**

This example creates a reference to an object named SUNLOGO, positioning it at the current position.

\*Include a Graphic.

C	MOVE	'SUNLOGO'	AFPOBNAM
C	Z-ADD	PAGESEGOT	AFPOBTYPE
C	Z-ADD	2880	AFPOBXSIZ
C	Z-ADD	2880	AFPOBYSIZ
C	Z-ADD	DEG0	AFPOBRTAT

## Include Object

C	Z-ADD	POSDEFLT	AFPOBMQPT
C	Z-ADD	0	AFPOBXPOS
C	Z-ADD	0	AFPOBYPOS
C	EXSR	AFPINOBJR	

## Include Overlay

### Function

Creates a reference to an overlay at the current position. You can include up to 127 unique page overlays on a page.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPInc10vly(
    TBHANDLE    in_pageHandle,
    char*        in_overlayName
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPIOVL.
    CALL "CBLInc10vly"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
            BY CONTENT
                AFP-OVLY-NAME
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLInc10vly" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

#### For the AFP Toolbox RPG Interface:

```
DAFPINCOVL      PR          10I 0 EXTPROC('AFPInc10vly')
D AFPPAGHDL      *          *  VALUE
D AFPOVNAM      9
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Overlay name

The name of the page overlay in the overlay library used by PSF. The overlay must be available to the printer at the time of printing. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, \_ (underscore), #, and @ (for example, O1MEDF01).

## Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

#### • For the AFP Toolbox C Library:

Tell PSF to include this overlay at print time.

```
OvlyName = "O1APQS1";
rc = AFPInc10vly(PageHandle, OvlyName);
```

#### • For the AFP Toolbox COBOL Interface:

Tell PSF to include this overlay at print time.

## Include Overlay

```
* IncludeOverlay
      MOVE "MYOVLY" TO AFP-OVLY-NAME.
      PERFORM AFPIOVL.
```

- **For the AFP Toolbox RPG Interface:**

Tell PSF to include this overlay at print time.

```
C          MOVE      'SEEDINV1'  AFPOVNAM
C          EXSR      AFPINOVLR
```



## Include Page Segment

### Function

Creates a reference to a page segment at the current location.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPIncIPSeg(
    TBHANDLE    in_pageHandle,
    char*        in_pageSegmentName,
    boolean      in_reuseOption
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPIPSEG.
    CALL "CBLIncIPSeg"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
            BY CONTENT
                AFP-PSEG-NAME
            BY VALUE
                AFP-REUSE-OPTION
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLIncIPSeg" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPINCPSG      PR          10I 0 EXTPROC('AFPIncIPSeg')
D AFPPAGHDL      *          VALUE
D AFPPSNAM       9
D AFPREUSE       10I 0 VALUE
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Page segment name

The name of the page segment in the page segment library. The page segment must be available to the print server at print time, but AFP Toolbox does not verify its existence. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, \_ (underscore), #, and @.

### Reuse option

Indicates that the referenced page segment is to reuse on multiple pages within the document.

## Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

### • For the AFP Toolbox C Library:

Tell PSF to include this page segment at print time and that it is reused multiple times in the document.

## Include Page Segment

```
PsegName = "S1LOG01";  
rc = AFPIncIPSeg(PageHandle, PsegName, TRUE);
```

- **For the AFP Toolbox COBOL Interface:**

This example writes the Include Page Segment (IPS) structured field into the output data stream, instructing the presentation system to include a page segment named MYPSEG at the current position.

```
* IncludePSeg  
      MOVE "MYPSEG" TO AFP-PSEG-NAME.  
      PERFORM AFPIPSEG.
```

- **For the AFP Toolbox RPG Interface:**

This example writes the Include Page Segment (IPS) structured field into the output data stream, instructing the presentation system to include a page segment named STRWNB at the current position.

```
*Sets the new position for printing a page segment.  
*at position 5 inches across and 8 inches down.  
C          Z-ADD      7200      AFPXCRD  
C          Z-ADD      11520     AFPYCRD  
C          Z-ADD      ABSPOS     AFPXRFCRDS  
C          Z-ADD      ABSPOS     AFPXRFCRDS  
C          EXSR      AFPSPOSR  
*Include a Page Segment  
C          MOVE      'STRWNB'    AFPPSNAM  
C          Z-ADD      FALS       AFPREUSE  
C          EXSR      AFPINPSGR
```

## Invoke Medium Map

### Function

Builds a request in the output to switch copy groups in the current form definition. Copy groups control output characteristics such as input bin, duplex, and so on. For more information about form definitions and copy groups, refer to *IBM Page Printer Formatting Aid: User's Guide*.

**Note:** Invoke Medium Map should be specified between pages. If you specify the same medium map on more than one Invoke Medium Map call in a row, the second and subsequent requests are ignored. If you want to use IMM to force physical page ejects in the printer, you must have two copy groups (medium maps) in your form definition and alternate between the two. This is a PSF restriction.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPIInvokeMediumMap(
    TBHANDLE    in_documentHandle,
    TOKEN       in_copygroup
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPINVMM.
    CALL "CBLInvokeMediumMap"
        USING
            BY VALUE
                AFP-DOCUMENT-HANDLE
            BY CONTENT
                AFP-MEDIUM-MAP-NAME
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLInvokeMediumMap" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPINVMM      PR          10I 0  EXTPROC('AFPIInvokeMediumMap')
D  AFPDOCHDL          *    VALUE
D  AFPMMNAM           9
```

### Input Parameters

#### Document handle

The current document handle. Document handle is returned by Begin Document.

#### Copy group

The name of the copy group (medium map) to use for printing this and subsequent pages in the document.

**Note:** AFP Toolbox does not verify the existence of the copy group in the form definition and returns no errors if the copy group does not exist. However, errors might result at print time.

### Return Codes

- 3** The document handle is missing or is not valid. Make sure you have called Begin Document before issuing this call.

## Invoke Medium Map

### Sample Function Call

- **For the AFP Toolbox C Library:**

Select the copy group named “CG1” in the current form definition.

```
strcpy(CopyGroup,"CG1");  
rc = AFPIInvokeMediumMap(DocHandle, CopyGroup);
```

- **For the AFP Toolbox COBOL Interface:**

Select the copy group named “CG1” in the current form definition.

```
* InvokeMediumMap  
      MOVE "CG1" TO AFP-MEDIUM-MAP-NAME.  
      PERFORM AFPINVMM.
```

- **For the AFP Toolbox RPG Interface:**

Select the copy group named “CG1” in the current form definition.

```
C          MOVEL      'CG1'          AFPMMNAM  
C          EXSR       AFPINVMMR
```

## Keep Object

### Function

Identifies an object that is to be kept at the printer while the page is printing. This may increase printer performance when large or complex objects are included in a page.

**Note:** This function is only valid if the output is sent to an Infoprint Color 130 Plus printer.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPKeepObject(
    TBHANDLE      in_pageHandle,
    char*          in_objectName,
    ObjectType     in_objectType,
    OtherObjectType in_otherObjType
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPKOBJ.
    CALL "CBLKeepObj"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
            BY CONTENT
                AFP-OBJECT-NAME
            BY VALUE
                AFP-OBJECT-TYPE
                AFP-OTHER-OBJECT-TYPE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLKeepObj" TO AFP-ERRDATA.
    MOVE SPACES TO AFP-OBJECT-NAME(AFP-NAME-LENGTH:1)
    PERFORM CHKSUCC.
```

## Input Parameters

### Object name

The name of the object to be kept at the printer.

### Object type

The type of object that is to be kept at the printer. Valid values are:

**IMAGE\_OT (for C), IMAGE-OT (for COBOL), or IMAGEOT (for RPG)**

An image object with MO:DCA syntax as defined in *Image Object Content Architecture Reference*

**OTHER\_OBJECT\_DATA\_OT (for C and C++), OTHER-OBJECT-DATA-OT (for COBOL), or OTHEROT (for RPG)**

Object data to include that is not defined by an IBM presentation architecture.

### Other object type

Use this parameter to fully identify the object type when Object Type is set to Other. Values are:

Value	Description
EPS_00T (for C), EPS-00T (for COBOL)	EPS
TIFF_00T (for C), TIFF-00T (for COBOL)	TIFF
DIB_WIN_00T (for C), DIB-WIN-00T (for COBOL)	DIB, Windows version
DIB_OS2_00T (for C), DIB-OS2-00T (for COBOL)	DIB, OS/2 PM version
PCX_00T (for C), PCX-00T (for COBOL)	PCX

## Keep Object

GIF\_00T (for C), GIF-00T (for COBOL)

JFIF\_00T (for C), JFIF-00T (for COBOL)

PDF\_SP0\_00T (for C), PDF-SP0-00T (for COBOL)

PCL\_PO\_00T (for C), PCL-PO-00T (for COBOL)

EPS\_TRAN\_00T (for C), EPS-TRAN-00T (for COBOL)

PDF\_TRAN\_00T (for C), PDF-TRAN-00T (for COBOL)

GIF

JFIF

PDF Single Page Object

PCL Page Object

EPS with Transparency

PDF with Transparency

## Return Codes

- 23** The object type is not valid. Valid object types are image and other.
- 24** Object type is other, but the other object type is not valid. See the Other object type parameter for valid values.

## MaxiCode Bar Code Data

### Function

Adds data to a MaxiCode bar code object previously started with Begin Bar Code.

### Syntax

```
int AFPMaxiCode (TBHANDLE      in_bcocaHandle,
                ushort         in_bcsymxpos,
                ushort         in_bcsymypos,
                boolean        in_bce2a,
                boolean        in_bcescape,
                short          in_bcsymmode,
                short          in_bcseqcount,
                short          in_bcseqind,
                boolean        in_bczipper,
                uchar*         in_bcddata);
```

Figure 11. C syntax

```
AFPMCODE.
    CALL "CBLMaxiCode"
        USING
            BY VALUE
                AFP-BCOCA-HANDLE
                AFP-BCSYM-XPOS,
                AFP-BCSYM-YPOS,
                AFP-BC-E2A
                AFP-BC-ESCAPE
                AFP-BC-SYMMODE
                AFP-BC-SEQCOUNT
                AFP-BC-SEQIND
                AFP-BC-ZIPPER
            BY CONTENT
                AFP-BC-DATA
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLMaxiCode" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

Figure 12. COBOL syntax

## Input Parameters

### BCOCA Handle

The handle for the bar code object to contain this data. BCOCA handle is returned by the Begin Bar Code call.

### Symbol X and Y position

Specifies the X and Y offsets from the presentation space origin to the top left corner of the beginning of the bar code symbol. Valid range is from 1 to 32767 1440ths. Zero is **not** valid for X or Y.

### EBCDIC to ASCII translation

This flag indicates whether the bar code data should be converted from EBCDIC to ASCII encoding. In order to print correctly the data and macro must be encoded in ASCII code page 819. In general, if you are running on a host platform such as OS/390 or AS/400, your data is EBCDIC by default so you should specify "true" for this parameter. On a workstation platform you would normally specify "false".

### Ignore Escape Sequence

This flag indicates whether a backslash character (ASCII X'5C' or EBCDIC X'E0') within the bar code

## MaxiCode Bar Code Data

data should be treated as an escape sequence or not. Specify “false” if the backslash should be treated as an escape, and “true” if backslashes should be treated as character data.

### Symbol Mode

Provides more information about the intended use of the bar code symbol. See the *Bar Code Object Content Architecture Reference* for details. Valid values are:

- MODE2** Structured Carrier Message with numeric only postal code. This mode is designed for use in the transport industry for encoding the postal code, country code, and service class. For **MODE2** and **MODE3**, the bar code data should be structured as described in B.2.1 and B.3.1 of the *AIM International Symbolology Specification—MaxiCode*. The first part of the bar code data includes the postal code, country code, and service class, in that order, separated by the [GS] character.
- MODE3** Same as **MODE2** except the postal code can contain characters as well as numbers.
- MODE4** Standard Symbol. The symbol uses Enhanced Error Correction (EEC) for the Primary Message and Standard Error Correction (SEC) for the Secondary Message. This mode provides for a total of 93 codewords for data.
- MODE5** Full ECC Symbol. The symbol uses Enhanced Error Correction (EEC) for both the Primary and Secondary Message. This mode provides for a total of 77 codewords for data.
- MODE6** Reader Program, SEC. The symbol uses Enhanced Error Correction (EEC for the Primary Message and Standard Error Correction (SEC) for the Secondary Message. This mode provides for a total of 93 codewords for data. The data in the symbol is used to program the bar code reader system.

### Sequence Count and Indicator

MaxiCode bar code symbols can be logically linked together to encode large amounts of data. This is called a structured append sequence. The logically linked symbols can be printed separately and then logically recombined after they are scanned. From 2 to 8 MaxiCode symbols can be linked together. The Sequence Count specifies the number of symbols to be linked. The Sequence Indicator specifies for each bar code symbol where it fits logically into the sequence (i.e. a number from 1 to 8). The Sequence Indicator must be 1 if the Sequence Count is 0 or 1.

### Zipper Pattern

Indicates whether or not a vertical zipper-like test pattern and contrast block is to be printed to the right of the symbol. The zipper provides a quick visual check for printing distortions. If the bar code is rotated, the zipper and block are rotated along with the symbol. Specify “true” to have the zipper and block printed and “false” if they are not needed.

### Data

The string of characters to be encoded as bar code data. For MaxiCode bar codes, up to 93 alphanumeric characters or up to 138 numeric characters can be given per symbol depending on encoding overhead. For details refer to the symbology specification. The input data can be ASCII or EBCDIC; use the E2A parameter to indicate whether translation from EBCDIC codepage 500 to ASCII codepage 819 is required.

## Return Codes

- 1** The **outp\_bcHandle** value is null. You must specify a valid address for this parameter.
- 100** The bar code position is out of range. The valid range is from 1—32767.
- 103** The sequence count must be in the range 0—8.  
The sequence indicator must be in the range 0—8.
- 105** The symbol mode must be in the range 2—6.



## Sample Function Call

Create a Data Matrix bar code:

### C Programming Language

```
| rc = AFPBgnBarCode(PageH, "BCOCA 1", MM_2_U1440(70),
|      MM_2_U1440(80), DEG 0, MAXICODE_BC, 0x00, DEFAULT_BCFONT,
|      BLACK, DEFAULT_BCMODWIDTH, -1, 1, -1, &BCH);
| rc = AFPMxiCode(BCH, MM_2_U1440(1), MM_2_U1440(1), TRUE,
|      TRUE, MODE4, 1, 1, TRUE,
|      "THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG 0123456789");
| rc = AFPEndBarCode(PageH, &BCH);
```

### COBOL Programming Language

```
| MOVE "MAXICODE BARCODE" TO AFP-BCOCA-NAME.
| MOVE 70 TO AFP-AREA-XSIZE.
| MOVE 80 TO AFP-AREA-YSIZE.
| MOVE MAXICODE-BC TO AFP-BC-TYPE.
| MOVE 0 TO AFP-BC-MODIFIER.
| MOVE -1 TO AFP-BC-FONT-ID.
| MOVE DEG-0 TO AFP-OBJECT-ROTATION.
| PERFORM AFPBBCD.
| MOVE 1 TO AFP-BCSYM-XPOS.
| MOVE 1 TO AFP-BCSYM-YPOS.
| MOVE FALS TO AFP-BCHRI-PRESENT.
| MOVE TRU TO AFP-BC-E2A.
| MOVE TRU TO AFP-BC-ESCAPE.
| MOVE MODE4 TO AFP-BC-SYMMODE.
| MOVE 1 TO AFP-BC-SEQCOUNT.
| MOVE 1 TO AFP-BC-SEQIND.
| MOVE "the quick brown fox jumped over the lazy dog"
| TO AFP-BC-2DDATA.
| PERFORM AFPMXIBC.
| PERFORM AFPEBCD.
```

|

# Measure Double-Byte String

## Function

Measures the width of the specified string text in the given font and returns the width. The specified font must be available to AFP Toolbox or this call will fail. The Measure Double-Byte String function assumes that the input character string is encoded in the code set of the specified font. To use a different input code set, see “Translate And Measure Double-Byte String” on page 244.

You can use the Query function to obtain the font line spacing value.

**Note:** You do not need to measure every string that is placed in your document. Measuring a lot of strings has a negative impact on the performance of your program.

## Syntax

```
int AFPMeasureDbString(  
    FontID      in_fontID,  
    char*       in_inputString,  
    long*       outp_stringWidth  
    ushort      in_vinc,  
    ushort      in_cadj,  
    int*        outp_varspaces  
    int*        outp_icspaces,  
);
```

## Input Parameters

### Font ID

The font ID returned by the Define Font By Attributes or Define Font By Name call.

### Character string

The string of DBCS characters to measure. The text is assumed to be encoded in the code set of the defined font. The defined font must be found with Define Double-Byte Font By Attribute.

### Variable increment

The width of the space between words in the specified font measured in 1440ths of an inch. The default value is defined by the font. Specify 0 to use the default value.

### Extra intercharacter space

Amount of extra space added to each intercharacter space (measured in 1440ths of an inch) as the string is measured. Specify 0 to use the default value.

## Output Parameters

### String width

The width of the string in the specified font. String width is returned in units of 1440ths of an inch.

### Variable spaces

A count of the number of spaces between words in the line of text. If this value is returned as zero and there are spaces in your input string, ensure that your input string is encoded in the code page of the specified font. The encoding of a blank varies based on whether the code page is ASCII or EBCDIC.

### Intercharacter spaces

A count of the number of character spaces in the line of text.

## Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

## Sample Function Call

This example assumes you have already received the FontID from either the Define Font By Name or Define Font By Attribute call and defined it with the same code page as the default code page for your system.

- **For the AFP Toolbox C Library:**

```
int ic_sp, var_sp, rc;
char CharString[LENGTH];
long StringWidth;
strcpy(CharString, "SBCS<DBCS>SBCS");
    where <,> are SOSI in host codepages
           SBCS is a string of SBCS
           DBCS is a string of DBCS
```

```
rc = AFPMeasureDbString (FontID,
                        CharString,
                        0,
                        0,
                        &StringWidth,;
                        &ic_sp,;
                        &var_sp);
```

---

# Measure String

## Function

Measures the width of the specified string in the given font and returns the width. The specified font must be available to AFP Toolbox or this call will fail. The Measure String function assumes that the input character string is encoded in the code page of the specified font. To use a different input code page, see “Translate And Measure String” on page 246.

You can use the Query function to obtain the font line spacing value.

**Note:** You do not need to measure every string that is placed in your document. Measuring a lot of strings has a negative impact on the performance of your program.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPMeasureString(  
    FontID      in_fontID,  
    char*       in_inputString,  
    long*       outp_stringWidth,  
    ushort     in_vinc,  
    ushort     in_cadj,  
    int*        outp_varspaces,  
    int*        outp_icspaces  
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPMSTR.  
    CALL "CBLMeasureString"  
        USING  
            BY VALUE  
                AFP-FONT-ID  
            BY CONTENT  
                AFP-CHARACTER-STRING  
            BY REFERENCE  
                AFP-MEASURED-WIDTH  
            BY VALUE  
                AFP-EXTRA-VARSPACE  
                AFP-EXTRA-ICSPACE  
            BY REFERENCE  
                AFP-VARSPACES  
                AFP-ICSPACES  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLMeasureStr" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPMEASTR      PR          101 0  EXTPROC('AFPMeasureString')  
D  AFPFONTID          51 0  VALUE  
D  AFPCHRSTR         250  
D  AFPSTRWID         101 0  
D  AFPXTRVAR          51 0  VALUE  
D  AFPXTRICS          51 0  VALUE  
D  AFPVARCNT         101 0  
D  AFPICSCNT         101 0
```

## Input Parameters

### Font ID

The font ID returned by the Define Font By Attributes or Define Font By Name call.

### Character string

The string of characters to measure. The text is assumed to be encoded in the code page of the defined font. No translation is performed.

### Variable increment

The width of the space between words in the specified font measured in 1440ths of an inch. The default value is defined by the font. Specify 0 to use the default value.

### Extra intercharacter space

Amount of extra space added to each intercharacter space (measured in 1440ths of an inch) as the string is measured. Specify 0 to use the default value.

## Output Parameters

### String width

The width of the string in the specified font. String width is returned in units of 1440ths of an inch.

### Variable spaces

A count of the number of spaces between words in the line of text. If this value is returned as 0 and there are spaces in your input string, ensure that your input string is encoded in the code page of the specified font. The encoding of a blank varies based on whether the code page is ASCII or EBCDIC.

### Intercharacter spaces

A count of the number of character spaces in the line of text.

## Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

This example assumes you have already received the FontID from either the Define Font By Name or Define Font By Attribute call and defined it with the same code page as the default code page for your system.

```
int ic_sp,var_sp;rc;
char CharString[LENGTH];
long StringWidth;

strcpy(CharString,"This is the string to measure");
rc = AFPMeasureString(FontID, CharString, 0, 0, &StringWidth,;
                      &ic_sp, &var_sp);
```

- **For the AFP Toolbox COBOL Interface:**

This example assumes you have started font Times New Roman Latin1 12 point and defined it with the same code page as the default code page for your system. After the call, AFP-MEASURED-WIDTH holds 2718, AFP-VARSPACE holds 5 and AFP-ICSPACE holds 18.

```
* Rely on defaults for
* AFP-EXTRA-ICSPACE AFP-EXTRA-VARSPACE
  MOVE TNR12 TO AFP-FONT-ID.
  MOVE "This is the string to measure" TO AFP-CHARACTER-STRING.
  PERFORM AFPMSTR.
```

- **For the AFP Toolbox RPG Interface:**

This example assumes you have started font Times New Roman Latin1 12 point and defined it with the same code page as the default code page for your system. After the call, AFPSTRWID holds 2718, AFPVARCNT holds 5 and AFPICSCNT holds 18.

## Measure String

```
C      DTA(6)      CAT(P)  x'00':0    AFPCMSTR
C      EXSR       AFPMEASRR
...
This is the string to measure
```

## Next Line

### Function

Advance one line in the current font. The horizontal position is not changed by this call.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPNextLine(
    TBHANDLE    in_pageHandle
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPINLINE.
    CALL "CBLNextLine"
        USING
            BY VALUE
            AFP-PAGE-HANDLE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLNextLine" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPINLINE          PR              10I 0 EXTPROC('AFPNextLine')
D  AFPPAGHDL        *      VALUE
```

### Input Parameters

#### Page handle

The handle for the page. Page handle is returned by the Begin Page call.

### Return Codes

- 4** Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 13** The width or depth of the line from the current location is off the logical page. The position is moved, but an error might result when your document is printed.
- 14** This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.

### Sample Function Call

- **For the AFP Toolbox C Library:**

Vertically advances one line space in the current font.

```
xpos = MM_2_U1440(5);
AFPMove(currentpage, xpos);
AFPNextLine(currentpage);
```

- **For the AFP Toolbox COBOL Interface:**

This example advances vertically one line space in the current font.

```
PERFORM AFPINLINE.
```

- **For the AFP Toolbox RPG Interface:**

This example advances vertically one line space in the current font.

```
C              EXSR      AFPNLINER
```

## Output Comment

### Function

Lets users to put comments in the data stream for use with postprocessors or other user-defined applications.

**Note:** This call generates a NOP (No Operation) structured field in the output.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPOutputComment(
    TBHANDLE    in_currHandle,
    char*        in_commentString
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPCOMMENT.
    CALL "CBLOutputComment"
        USING
            BY VALUE
                AFP-CURRENT-HANDLE
            BY CONTENT
                AFP-COMMENT-STRING
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLOutputComment" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPCMMNT      PR          10I 0  EXTPROC('AFPOutputComment')
D AFPCURHDL          *      VALUE
D AFPCMSTR          250
```

### Input Parameters

#### Current handle

The handle for the current document or page returned from the Begin Document or the Begin Page call. If the current handle is for the document, the comment is placed between pages. Otherwise, it goes inside the current page.

#### Comment string

The string of characters placed in the output as a comment. The string of data is ignored by AFP presentation products (such as PSF and AFP Workbench).

### Return Codes

- 3** The document handle is missing or is not valid. Make sure you have called Begin Document before issuing this call.
- 4** Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

### Sample Function Call

- **For the AFP Toolbox C Library:**



```
strcpy(CharString, "This is my comment string");
rc = AFPOutputComment(PageHandle, CharString);
```

• **For the AFP Toolbox COBOL Interface:**

This example shows a document level comment and a page level comment.

```
PERFORM AFPBDOC.
* this NOP comes out before the first BPG
  SET AFP-CURRENT-HANDLE TO AFP-DOCUMENT-HANDLE.
  MOVE "THIS IS DOCUMENT NOP1" TO AFP-COMMENT-STRING.
  PERFORM AFPCOMMENT.

  PERFORM AFPBPAGE.

* This NOP comes out after the 1st BPG
  SET AFP-CURRENT-HANDLE TO AFP-PAGE-HANDLE.
  MOVE "THIS IS PAGE1 NOP1 AFTER THE BPG"
    TO AFP-COMMENT-STRING.
  PERFORM AFPCOMMENT.
* Put the rest of your stuff on the page
  PERFORM AFPEPAGE.
  PERFORM AFPEDOC.
```

• **For the AFP Toolbox RPG Interface:**

This example shows a document level comment and a page level comment.

```
C          EXSR      AFPBDOCR
C      DTA(6)      CAT(P)  x'00':0      AFPCMSTR
C          EXSR      AFPCMMNR
C          EXSR      AFPBPAGER
C      DTA(7)      CAT(P)  x'00':0      AFPCMSTR
C          EXSR      AFPCMMNR
...
Text for my document level comment      6
Text for my page level comment          7
```

## PDF417 Bar Code Data

### Function

Adds data to a PDF417 bar code object previously started with Begin Bar Code.

### Syntax

```
int AFPPDF417 (TBHANDLE      in_bcocaHandle,
               ushort        in_bcsymxpos,
               ushort        in_bcsymypos,
               short         in_bcnumrows,
               short         in_bcrowsize,
               short         in_bcsecllevel,
               boolean        in_bce2a,
               boolean        in_bcescape,
               uchar*         in_bcddata,
               uchar*         in_bcmacro);
```

Figure 13. C syntax

```
AFPP417.
    CALL "CBLPDF417"
        USING
        BY VALUE
            AFP-BCOCA-HANDLE
            AFP-BCSYM-XPOS,
            AFP-BCSYM-YPOS,
            AFP-BC-NUMROWS
            AFP-BC-ROWSIZE
            AFP-BC-SEC-LEVEL
            AFP-BC-E2A
            AFP-BC-ESCAPE
        BY CONTENT
            AFP-BC-DATA
            AFP-BC-MACRO
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLPDF417" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

Figure 14. COBOL syntax

### Input Parameters

#### BCOCA Handle

The handle for the bar code object to contain this data. BCOCA handle is returned by the Begin Bar Code call.

#### Symbol X and Y position

Specifies the X and Y offsets from the presentation space origin to the top left corner of the beginning of the bar code symbol. Valid range is from 1 to 32767 1440ths. Zero is **not** valid for X or Y.

#### Number of Rows

The desired number of rows in the generated bar code symbol. From 3 to 90 rows can be requested, or specify 0 as the number of rows to have the printer generate the minimum number of rows necessary. The number of rows times the number of data symbol characters per row cannot exceed 928. The actual number of rows generated by the printer depends on the amount of data and the

security level selected. If more rows are requested with this parameter than are necessary, the symbol is padded to fill the requested number. If not enough rows are specified, extra rows will be inserted by the printer to produce the symbol.

### Row Size

The number of data symbol characters per row. Each row consists of a start pattern, a left row indicator codeword, 1 to 30 data symbol characters, a right row indicator codeword, and a stop pattern. The number of rows times the number of data symbol characters per row cannot exceed 928.

### Security Level

The desired security level for the symbol as an integer from 0 to 8. The higher the security level, the more error correction codewords will be added to the symbol. The following list shows a recommended security level for various amounts of data:

Number of Data Codewords	Recommended Security Level
1—40	2
41—160	3
161—320	4
321—863	5

### EBCDIC to ASCII translation

This flag indicates whether the bar code data and macro should be converted from EBCDIC to ASCII encoding. In order to print correctly the data and macro must be encoded in the default character encodation (GLI 0). For details see the *Bar Code Object Content Architecture Reference*. In general, if you are running on a host platform such as OS/390 or AS/400, your data is EBCDIC by default so you should specify “true” for this parameter. On a workstation platform you would normally specify “false”.

### Ignore Escape Sequence

This flag indicates whether a backslash character (ASCII X'5C' or EBCDIC X'E0') within the bar code data should be treated as an escape sequence or not. Specify “false” if the backslash should be treated as an escape, and “true” if backslashes should be treated as character data.

### Data

The string of characters to be encoded as bar code data. For PDF417 bar codes, up to 1850 text characters, 2710 ASCII numeric digits, or 1108 bytes of binary data per symbol are allowed, depending on the security level. For details refer to the symbology specification. The input data can be ASCII or EBCDIC; use the E2A parameter to indicate whether translation by the printer is required.

### Macro

This parameter consists of a Macro PDF417 Control Block. The data for this macro must adhere to the format defined in section G.2 of the Uniform Symbology Specification PDF417. The AFP Toolbox will not attempt to verify the macro contents so you may end up with errors at print time. The macro data can be ASCII or EBCDIC; use the E2A parameter to indicate whether translation by the printer is required. Specify NULL for this parameter if there is no macro data for the bar code.

## Return Codes

- 1** The **outp\_bcHandle** value is null. You must specify a valid address for this parameter.
- 100** The bar code position is out of range. The valid range is from 1—32767.
- 101** The number of rows must be 0 or in the range 3—90.  
The number of data symbols per row must be from 1—30.  
The product of the row size times the number of rows is greater than 928.
- 102** The security level is greater than 8.

## PDF417 Bar Code Data

### Sample Function Call

Create a PDF417 bar code:

#### C Programming Language

```
| rc = AFPBgnBarCode(hPage, "BCOCA 1", MM_2_U1440(120),  
| MM_2_U1440(120), DEG_0, PDF417_BC, 0x00, DEFAULT_BCFONT,  
| BLACK, DEFAULT_BCMODWIDTH, -1, 1, -1,  
| &hBcoca);  
|  
| rc = AFPPDF417(hBcoca,1,1,3,5,0,TRUE,TRUE,  
| "The quick brown fox jumped over the lazy dogs 0123456789",  
| NULL);  
|  
| rc = AFPEndBarCode(hPage, &hBcoca);
```

#### COBOL Programming Language

```
| MOVE "PDF BARCODE" TO AFP-BCOCA-NAME.  
| MOVE 120 TO AFP-AREA-XSIZE.  
| MOVE 120 TO AFP-AREA-YSIZE.  
| MOVE PDF417-BC TO AFP-BC-TYPE.  
| MOVE 0 TO AFP-BC-MODIFIER.  
| MOVE -1 TO AFP-BC-FONT-ID.  
| MOVE DEG-0 TO AFP-OBJECT-ROTATION.  
| PERFORM AFPBBCD.  
| MOVE 1 TO AFP-BCSYM-XPOS.  
| MOVE 1 TO AFP-BCSYM-YPOS.  
| MOVE 0 TO AFP-BC-NUMROWS.  
| MOVE 5 TO AFP-BC-ROWSIZE.  
| MOVE 2 TO AFP-BC-SECLEVEL.  
| MOVE FALS TO AFP-BCHRI-PRESENT.  
| MOVE TRU TO AFP-BC-E2A.  
| MOVE TRU TO AFP-BC-ESCAPE.  
| MOVE "bc data" TO AFP-BC-2DDATA.  
| MOVE "macro stuff" TO AFP-BC-2DMACRO.  
| PERFORM AFPPDFBC.  
| PERFORM AFPEBCD.  
  
|
```

## Preload Object

### Function

Identifies an object that is to be sent to the printer before printing. This may increase printer performance when large or complex objects are included in a document.

**Note:** This function is only valid if the output is sent to an Infoprint Color 130 Plus printer.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPPreloadObject(
    TBHANDLE      in_docHandle,
    char*          in_objectName,
    ObjectType     in_objectType ,
    OtherObjectType in_otherObjType
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPPOBJ.
    CALL    "CBLPre1Obj"
           USING
           BY VALUE
           AFP-DOCUMENT-HANDLE
           BY CONTENT
           AFP-OBJECT-NAME
           BY VALUE
           AFP-OBJECT-TYPE
           AFP-OTHER-OBJECT-TYPE
           RETURNING
           AFP-RET-CODE.
MOVE "CBLPre1Obj" TO AFP-ERRDATA.
MOVE SPACES TO AFP-OBJECT-NAME(AFP-NAME-LENGTH:1)
PERFORM CHKSUCC.
```

## Input Parameters

### Object name

The name of the object to be sent to the printer before printing.

### Object type

The type of object that is to be sent to the printer before printing. Values are:

**IMAGE\_OT (for C), IMAGE-OT (for COBOL), or IMAGEOT (for RPG)**

An image object with MO:DCA syntax as defined in *Image Object Content Architecture Reference*

**OTHER\_OBJECT\_DATA\_OT (for C and C++), OTHER-OBJECT-DATA-OT (for COBOL), or OTHEROT (for RPG)**

Object data to include that is not defined by an IBM presentation architecture.

### Other object type

Use this parameter to fully identify the object type when Object Type is set to Other. Values are:

Value	Description
EPS_00T (for C), EPS-00T (for COBOL)	EPS
TIFF_00T (for C), TIFF-00T (for COBOL)	TIFF
DIB_WIN_00T (for C), DIB-WIN-00T (for COBOL)	DIB, Windows version
DIB_OS2_00T (for C), DIB-OS2-00T (for COBOL)	DIB, OS/2 PM version
PCX_00T (for C), PCX-00T (for COBOL)	PCX

## Preload Object

GIF\_00T (for C), GIF-00T (for COBOL)

JFIF\_00T (for C), JFIF-00T (for COBOL)

PDF\_SP0\_00T (for C), PDF-SP0-00T (for COBOL)

PCL\_PO\_00T (for C), PCL-P0-00T (for COBOL)

EPS\_TRAN\_00T (for C), EPS-TRAN-00T (for COBOL)

PDF\_TRAN\_00T (for C), PDF-TRAN-00T (for COBOL)

GIF

JFIF

PDF Single Page Object

PCL Page Object

EPS with Transparency

PDF with Transparency

## Return Codes

**23** The object type is not valid. Valid object types are image and other.

**24** Object type is other, but the other object type is not valid. See the Other object type parameter for valid values.

---

## Preload Overlay

### Function

Identifies an overlay that is to be sent to the printer before printing. This may increase printer performance when large or complex overlays are included in a document.

**Note:** This function is only valid if the output is sent to an Infoprint Color 130 Plus printer.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPPreloadOverlay(  
    TBHANDLE      in_docHandle,  
    char*          in_overlayName,  
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPPOVL.  
    CALL "CBLPre10vly"  
        USING  
            BY VALUE  
                AFP-DOCUMENT-HANDLE  
            BY CONTENT  
                AFP-OVLY-NAME  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLPre10vly" TO AFP-ERRDATA.  
    MOVE SPACES TO AFP-OVLY-NAME(AFP-NAME-LENGTH:1)  
    PERFORM CHKSUC.
```

### Input Parameters

#### Overlay name

The name of the object to be kept at the printer.

---

## Put Box

### Function

Draws a box from the current position extending in the requested x and y directions using the specified rule thickness. The box must be ended before you end the page. Ensure that the box you have specified fits on the logical page. Do not issue this call within a paragraph or a table.

**Note:** To provide device-independent formatting, the output data stream produced by AFP Toolbox has a resolution of 1440 ppi. The printer rounds positioning information from 1440ths of an inch to the nearest pel (240, 300, and so forth). This might cause one pel rounding errors that could be evident in rule widths and rule intersections.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPPutBox(  
    TBHANDLE    in_pageHandle,  
    short        in_rulethickness,  
    short        in_boxwidth,  
    short        in_boxdepth  
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPPBOX.  
    CALL "CBLPutBox"  
        USING  
            BY VALUE  
                AFP-PAGE-HANDLE  
            BY VALUE  
                AFP-RULE-THICKNESS  
                AFP-BOX-WIDTH  
                AFP-BOX-DEPTH  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLPutBox" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

#### For the AFP Toolbox RPG Interface:

```
DAFPPUTBOX      PR          10I 0  EXTPROC('AFPPutBox')  
D  AFPPAGHDL          *  VALUE  
D  AFPRULTHK          10I 0  VALUE  
D  AFPBOXWID          5I 0  VALUE  
D  AFPBOXDEP          5I 0  VALUE
```

### Input Parameters

#### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

#### Rule thickness

The thickness of the rule to use to draw the box.

#### Box width

Width of the box. The width of the box is calculated from the left side of the left most rule to the left side of the right most rule. In other words, the box width includes the width of the left side rule but not the right side rule. If the width is a positive number, the box is drawn to the right of the current position; otherwise it is drawn to the left of the current position.



**Box depth**

Depth of the box. The depth of the box is calculated from the top of the topmost rule to the top of the bottommost rule. In other words, the box depth includes the width of the top rule but not the bottom rule. If the depth is a positive number, the box is drawn down from the current position; otherwise it is drawn up from the current position.

**Return Codes**

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 14 This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.

**Sample Function Call**

- **For the AFP Toolbox C Library:**

Draw a box one inch square using a rule that is 1/10th inch thick:

```
boxrule = IN_2_U1440(.10);
boxside = IN_2_U1440(1);
AFPPutBox(PageHandle, boxrule, boxside, boxside);
```

- **For the AFP Toolbox COBOL Interface:**

This example moves the current position to one inch across, one inch down and draws a box one inch wide by one inch deep with 50 pel thick rules.

```
MOVE 1440 TO AFP-X-COORDINATE.
MOVE 1440 TO AFP-Y-COORDINATE.
MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
PERFORM AFPSPOS.
MOVE 50 TO AFP-RULE-THICKNESS.
MOVE 1440 TO AFP-BOX-WIDTH.
MOVE 1440 TO AFP-BOX-DEPTH.
PERFORM AFPPBOX.
```

- **For the AFP Toolbox RPG Interface:**

This example moves the current position to one inch across, one inch down, and draws a box one inch wide by one inch deep with 50 pel thick rules.

C	Z-ADD	50	AFPRULTHK
C	Z-ADD	1440	AFPPBOXWID
C	Z-ADD	1440	AFPPBOXDEP
C	EXSR	AFPPBOXR	

---

## Put Color

### Function

Creates a rectangular block of color at the current position.

**Note:** Use of this function requires microcode support in your printer.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPPutColor(  
    TBHANDLE    in_pageHandle,  
    ColorSpace  in_coltype,  
    MODCAColors in_color,  
    ulong       in_colspec1,  
    ulong       in_colspec2,  
    ulong       in_colspec3,  
    ulong       in_colspec4,  
    ulong       in_colwidth,  
    ulong       in_coldepth  
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPPCLR.  
    CALL      "CBLPutColor"  
            USING  
            BY VALUE  
            AFP-PAGE-HANDLE  
            AFP-COLOR-TYPE  
            AFP-COLOR  
            AFP-COLSPEC1  
            AFP-COLSPEC2  
            AFP-COLSPEC3  
            AFP-COLSPEC4  
            AFP-COLOR-WIDTH  
            AFP-COLOR-DEPTH  
            RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLPutColor" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. Page handle is returned by the Begin Graphic call.

#### Color Space Type

Specifies the type of color to generate:

##### RGB\_COLOR (for C) or RGB-COLOR (for COBOL)

The color value is specified as three intensity levels for the red, green, and blue components of the color. These numbers are given for color specification values 1, 2, and 3. The fourth specification is ignored and should be specified as 0.

##### CMYK\_COLOR (for C) or CMYK-COLOR (for COBOL)

The color value is specified as four intensity levels for the cyan, magenta, yellow, and black components of the color. These are color specifications 1-4.

**HIGHLIGHT\_COLOR (for C) or HIGHLIGHT-COLOR (for COBOL)**

Defines a request for presentation device to generate a highlight color. Specify a MO:DCA Color value for the highlight color. Color specification 1 is the percentage coverage for the specified color, and color specification 2 is the percentage of black that is added to the coverage (can be 0). Color specifications 3 and 4 should be 0.

**CIELAB\_COLOR (for C) or CIELAB-COLOR (for COBOL)**

Specified with three components. Color specification 1 is the luminance, and color specifications 2 and 3 are the chrominance differences. Color specification 4 should be 0.

**OCA\_COLOR (for C) or OCA-COLOR (for COBOL)**

Defines a request for presentation device to generate a highlight color. Specify a MO:DCA Color value for the highlight color. The presentation device will produce printed colors using a color mapping table (see PSF documentation for your environment). Color specifications 1-4 should be 0.

**MODCA Color**

Any of the defined MO:DCA color values: BLUE, RED, MAGENTA, GREEN, CYAN, YELLOW, BLACK, and BROWN. This value is only used for Highlight or OCA color spaces but always must be specified due to C language conventions.

**Color specifications 1-4**

Usage depends on the color space type, see descriptions above.

**Color width and depth**

The dimensions of the color rectangle created.

**Return Codes**

none

**Sample Function Call**

Create a colored rectangle with a blue highlight color space, 75 percent blue coverage and 25 percent black.

- **For the AFP Toolbox C Library:**

```
rc = AFPPutColor(pageHandle,HIGHLIGHT_CS,BLUE,75,25,0,0,2000,1440);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE HIGHLIGHT-CS TO AFP-COLOR-TYPE.
MOVE 75 TO AFP-COLSPEC1.
MOVE 25 TO AFP-COLSPEC2.
MOVE 0 TO AFP-COLSPEC3.
MOVE 0 TO AFP-COLSPEC4.
MOVE 2000 TO AFP-COLOR-WIDTH.
MOVE 1440 TO AFP-COLOR-DEPTH.
MOVE BLUE TO AFP-COLOR.
PERFORM AFPPCLR.
```

# Put Double-Byte Text

## Function

Add a string of a mixture of SBCS and DBCS text to the current paragraph, using the formatting characteristics specified on the Begin Paragraph call. The text must be ended before you end the page. Ensure that the text you have specified fits on the logical page.

**Note:** There is a limit of 132 characters per word in the string of text.

## Syntax

**For the AFP Toolbox C Library:**

```
int AFPPutDbText (
    TBHANDLE    in_pageHandle,
    char*        in_textstring,
    FontID       in_textfont,
    char*        outp_remainstr
);
```

**For the AFP Toolbox COBOL Library:**

```
AFPPDBTXT.
CALL "CBLPutDbText"
    USING
    BY VALUE
        AFP-PAGE-HANDLE
    BY CONTENT
        AFP-CHARACTER-STRING
    BY VALUE
        AFP-FONT-ID
    BY REFERENCE
        AFP-REMAINING-STRING
    RETURNING
        AFP-RET-CODE.
MOVE "CBLPutDbText" TO AFP-ERRDATA.
PERFORM CHKSUCC.
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Text string

The null-terminated string of characters placed.

### Text font

The font for the text string. Text font is returned by Define Double-Byte Font By Attribute.

## Output Parameters

### Remaining text

The string of DBCS characters that would not fit in the paragraph.

## Return Codes

- 4** Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 8** Font not defined. You must successfully define the font with Define Font By Name or Define Font By Attribute before issuing Set Font.

- 12 The entire string does not fit in the paragraph. The portion that did not fit is returned in the “remaining text” variable.
- 18 A single word was too wide to fit in the paragraph. The **outp\_remainstr** parameter points to the word that did not fit. Processing of the input text string ends preceding this long word.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Create a paragraph 125mm wide by up to 25mm deep with a left-justified DBSC string of text and place that text in the paragraph.

```
#include "aocl.h"

TBHANDLE currentdoc;
TBHANDLE currentpage;
TBHANDLE currentsess;

rc = AFPBgnSession(&currentsess); // start the session
rc = AFPBgnDoc(currentsess, docname, doccomment,
               TO_OUTPUT_FILE,
               myfile, mybuff,
               &currentdoc);    // start the document

rc = AFPSetFontType(FONT_OLN);

rc = AFPSetInputDbCodeset(currentsess, "IBM-942", ' ');

rc = AFPDefineDbFontByAttr("T1H00290", "T10300",
                          "IBM-939",
                          "HEISEI GOTHIC",
                          MEDIUM_WD, NORMAL_FS, &f1);

rc = AFPBgnGroup(currentdoc, "Group 1");

rc = AFPBgnPage(currentdoc, "Page 1", &currentpage);

AFPSetTextOrientation(currentpage, I0B90_T0);

AFPSetFont(currentpage, f1);
AFPSetPos(currentpage, 240, ABSOLUTE_POS, 4*240, ABSOLUTE_POS);
AFPBgnPgraph(currentpage, MM_2_U1440(125), JUSTIFY_ALIGN, DEFAULT_LSP,
             MM_2_U1440(25));

AFPPutDbText(currentpage,
             "This paragraph is justify-aligned.",
             f1,
             &remainingstr);

AFPEndPgraph(currentpage);
rc = AFPEndPage(&currentpage);
```

- **For the AFP Toolbox COBOL Interface:**

This example assumes you have used Define Double-Byte Font by Attribute to define Heisei Gothic as F1.

```
MOVE "This paragraph is justify-aligned." TO AFP-CHARACTER-STRING.
MOVE F1 TO AFP-FONT-ID.
PERFORM AFPPDBTXT.
```

See “Begin Paragraph” on page 93 for a sample paragraph of text.

---

# Put Horizontal Rule

## Function

Draws a rule from the current position extending in the x direction using the specified rule thickness. The rule thickness extends below horizontal rules (that is, the current position is at the top left corner of the rule). The rule must be ended before you end the page. Ensure that the rule you have specified fits on the logical page. Do not issue this call within a paragraph or a table.

**Note:** To provide device-independent formatting, the output data stream produced by AFP Toolbox has a resolution of 1440 ppi. The printer rounds positioning information from 1440ths of an inch to the nearest pel (240, 300, and so forth). This might cause one pel rounding errors that could be evident in rule widths and rule intersections.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPPutHRule(
    TBHANDLE    in_pageHandle,
    short        in_ruleLength,
    short        in_ruleThickness
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPPHRUL.
    CALL "CBLPutHRule"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-RULE-LENGTH
                AFP-RULE-THICKNESS
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLPutHRule" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPPUTHRL      PR              10I 0 EXTPROC('AFPPutHRule')
D AFPPAGHDL          *  VALUE
D AFPRULLEN         10I 0  VALUE
D AFPRULTHK         10I 0  VALUE
```

## Input Parameters

### Page handle

The handle for the current page. Page handle returned by the Begin Page call.

### Rule length

The length of the rule. If the length is negative, the rule is drawn to the left of the current position; otherwise it is drawn to the right of the current position.

### Rule thickness

The thickness of the rule.

## Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

- 14 This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Draw a one inch horizontal rule to the right of the current location using a rule thickness of 1/10th inch.

```
rulethick = IN_2_U1440(.10);
rulelength = IN_2_U1440(1);
rc = AFPPutHRule(PageHandle, rulelength, rulethick);
```

- **For the AFP Toolbox COBOL Interface:**

This example draws a horizontal rule starting at the current position for three inches. The rule is .25 inch thick.

```
*3 inch  horiz rule at current position
      MOVE 4320 TO AFP-RULE-LENGTH.
      MOVE 360  TO AFP-RULE-THICKNESS.
      PERFORM AFPPHRUL.
```

- **For the AFP Toolbox RPG Interface:**

This example draws a horizontal rule starting at the current position for three inches. The rule is .25 inch thick.

```
* 3 inch  horiz rule at current position
```

# Put Shade

## Function

Puts a shaded rectangle at the current location.

**Note:** Use of this function requires microcode support in your printer.

## Syntax

**For the AFP Toolbox C Library:**

```
int AFPPutShade(
    TBHANDLE    in_pageHandle,
    ushort      in_shade_percent,
    ushort      in_shade_width,
    ushort      in_shade_depth
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPPSHD
    CALL "CBLPutShade"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-SHADE-PERCENT
                AFP-SHADE-WIDTH
                AFP-SHADE-DEPTH
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLPutShade" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

## Input Parameters

### Page handle

The handle for this page. Page handle is returned by the Begin Shade call.

### Shade percent

Specifies how much shading is applied to the shaded area. The valid values are integers from 0 to 100.

### Shade width and depth

The width and depth of the shaded area.

## Return Codes

- 4      Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

Create a shaded rectangle three inches square shaded 25 percent:

- **For the AFP Toolbox C Library:**

```
AFPPutShade (PageHandle,25,4320,4320);
```

- **For the AFP Toolbox COBOL Interface:**

```
    MOVE 4320 TO AFP-SHADE-WIDTH.
    MOVE 4320 TO AFP-SHADE-DEPTH.
    MOVE 25 TO AFP-SHADE-PERCENT.
    PERFORM AFPPSHD.
```



## Put Tag

### Function

Creates an indexing tag in the document for use by presentation systems such as the AFP Workbench or postprocessor applications such as OnDemand. This call generates a Tag Logical Element (TLE) structured field at either the group or the page level depending on which handle is specified.

**Note:** If you issue an Invoke Medium Map call in your document, any subsequent Put Tag requests at the document level are ignored.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPPutTag (
    TBHANDLE    in_currentHandle,
    char*       in_attrName,
    char*       in_attrValue
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPPTAG.
    CALL "CBLPutTag"
        USING
            BY VALUE
                AFP-CURRENT-HANDLE
            BY CONTENT
                AFP-TAG-NAME
                AFP-TAG-VALUE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLPutTag" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPPUTTAG      PR              10I 0  EXTPROC('AFPPutTag')
D AFPCURHDL          *  VALUE
D AFPTAGNAM          250
D AFPTAGVAL          250
```

## Input Parameters

### Current handle

The handle for the current document or page returned from the Begin Document or the Begin Page call. If the current handle is the document, a group level tag is built. Otherwise, a page level tag is built.

### Attribute name

The name of the indexing attribute, for example, "Policy Number". The maximum number of characters in the attribute name is 250, including blanks. If more than 250 characters are specified, the string is truncated.

### Attribute value

The value of the indexing attribute, for example, "4327894-8324". The maximum number of characters in the attribute value is 250, including blanks. If more than 250 characters are specified, the string is truncated.

## Put Tag

### Return Codes

- 1 Current handle is missing or is not valid. Make sure you have called Begin Document before issuing this call.
- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

### Sample Function Call

- **For the AFP Toolbox C Library:**

Tag this page with the attribute named "Policy Number" and the value for this page:

```
strcpy(AttrName, "Policy Number");  
strcpy(AttrValue, "4327894-8324");  
rc = AFPPutTag(PageHandle, AttrName, AttrValue);
```

- **For the AFP Toolbox COBOL Interface:**

See "Begin Group" on page 84 for a complete Put Tag example.

- **For the AFP Toolbox RPG Interface:**

See "Begin Group" on page 84 for a complete Put Tag example.

## Put Text

### Function

Add a string of text to the current paragraph, using the formatting characteristics specified on the Begin Paragraph call. The text must be ended before you end the page. Ensure that the text you have specified fits on the logical page.

#### Notes:

1. There is a limit of 132 characters per word in the string of text.
2. Do not use this call to put text in a table. Use the 'Put Text in Table Field' function call instead. See 205 for details.

#### Note:

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPPutText (
    TBHANDLE    in_pageHandle,
    char*        in_textstring,
    FontID       in_textfont,
    char*        outp_remainstr
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPPTXT.
    CALL "CBLPutText"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
            BY CONTENT
                AFP-CHARACTER-STRING
            BY VALUE
                AFP-FONT-ID
            BY REFERENCE
                AFP-REMAINING-STRING
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLPutText" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**Note:** You must set the length of the string in AFP-STRING-LENGTH before calling this function. You can use the TRIM and STRINGLEN functions provided in MVS and OS/400 to calculate this length. See the appendix for your operating system for more information.

#### For the AFP Toolbox RPG Interface:

```
DAFPPTXT      PR          10I 0 EXTPROC('AFPPutText')
D AFPPAGHDL          *    VALUE
D AFPCHRSTR          250
D AFPFONTID          5I 0 VALUE
D AFPREMSTR          *
```

### Input Parameters

#### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

## Put Text

### Text string

The null-terminated string of characters to place.

### Text font

The font for the text string. Text font is returned by Define Font By Attribute or Define Font By Name.

## Output Parameters

### Remaining text

The string of characters that would not fit in the paragraph.

## Return Codes

- 4      Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 8      Font not defined. You must successfully define the font with Define Font By Name or Define Font By Attribute before issuing Set Font.
- 12     The entire string does not fit in the paragraph. The portion that did not fit is returned in the “remaining text” variable.
- 18     A single word was too wide to fit in the paragraph. The **outp\_remainstr** parameter points to the word that did not fit. Processing of the input text string ends preceding this long word.

## Sample Function Call

See “Begin Paragraph” on page 93 for a sample paragraph of text.

## Put Text in Table Field

### Function

Add a string of text to the current field in a table. You must issue Begin Row and Begin Field before issuing this call.

### Syntax

#### For the AFP Toolbox C Library:

```
int AFPPutTextInTableFld(
    TBHANDLE in_pageHandle,
    uint in_fielddid,
    char* in_textstring,
    FontID in_textfont
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPPCHS.
    CALL "CBLPutTextInTableFld"
    USING
        BY VALUE
            AFP-PAGE-HANDLE
            AFP-FIELD-ID
        BY CONTENT
            AFP-CHARACTER-STRING
        BY VALUE
            AFP-FONT-ID
    RETURNING
        AFP-RET-CODE.
    MOVE "CBLPutTextInTableFld" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### Page handle

The handle for this page. Page handle is returned by Begin Page.

#### Field ID

The ID of the field to add text to.

#### Character string

The string to add to the table.

#### Font ID

The font identifier returned from Define Font.

### Return Codes

- 8** The font ID parameter is invalid. You must successfully define the font with Define Font By Name or Define Font By Attribute before issuing Set Font.
- 26** The row object handle is invalid or null. You must issue Begin Row before issuing Put Text in Table Field.
- 41** The table object handle is invalid or null. You must issue Begin Table before issuing Put Text in Table Field.
- 63** The text string length is invalid. You must use Put Text In Table Field to put text in a table. If you used that call, make sure you entered a non-zero length string for the text string.

## Put Text in Table Field

**64** The text string to enter in the table is too long. The string cannot exceed the value specified by TABLEXTSIZE (This is 250 bytes for COBOL).

## Sample Function Call

See “Begin Field” on page 82 for a more complete example.

- **For the AFP Toolbox C Library:**

```
AFPPutTextInTableFld(in_pageHandle, in_fldid, in_textstring, in_textfont);
```

- **For the AFP Toolbox COBOL Interface:**

```
    MOVE "User defined string" TO AFP-CHARACTER-STRING.  
    MOVE FONT-ID TO AFP-FONT-ID  
    PERFORM AFPPCHS.
```

## Put Vertical Rule

### Function

Draws a rule from the current position extending in the y direction using the specified rule thickness. The rule thickness extends to the right of vertical rules (the current position is at the top left corner of the rule or the bottom left corner if the length is negative). The rule must be ended before you end the page. Ensure that the rule you have specified fits on the logical page.

**Note:** To provide device-independent formatting, the output data stream produced by AFP Toolbox has a resolution of 1440 ppi. The printer rounds positioning information from 1440ths of an inch to the nearest pel (240, 300, and so forth). This might cause one pel rounding errors that could be evident in rule widths and rule intersections.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPPutVRule(
    TBHANDLE    in_pageHandle,
    short        in_ruleLength,
    short        in_ruleThickness
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPPVRUL.
    CALL "CBLPutVRule"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-RULE-LENGTH
                AFP-RULE-THICKNESS
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLPutVRule" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPPUTVRL      PR              10I 0 EXTPROC('AFPPutVRule')
D AFPPAGHDL      *              VALUE
D AFPRULLEN      10I 0          VALUE
D AFPRULTHK      10I 0          VALUE
```

### Input Parameters

#### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

#### Rule length

The length of the rule. If the length is negative, the rule extends up from the current position; otherwise it is drawn down from the current position.

#### Rule thickness

The thickness of the rule.

### Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Put Vertical Rule

### Sample Function Call

- **For the AFP Toolbox C Library:**

Draw a one inch vertical rule down from the current location, using a rule thickness of 1/10th inch.

```
rulethick = IN_2_U1440(.10);  
rulelength = IN_2_U1440(1);  
rc = AFPPutVRule(PageHandle, rulelength, rulethick);
```

- **For the AFP Toolbox COBOL Interface:**

Draw a one inch vertical rule down from the current location, using a rule thickness of 1/10th inch.

```
MOVE 1440 TO AFP-RULE-LENGTH.  
MOVE 144 TO AFP-RULE-THICKNESS.  
PERFORM AFPPVRUL.
```

- **For the AFP Toolbox RPG Interface:**

Draw a one inch vertical rule down from the current location, using a rule thickness of 1/10th inch.

```
C           MOVE      1440      AFPRULLEN  
C           MOVE      144      AFPRULTHK  
C           EXSR      AFPPVRLR
```



## Query

### Function

Returns the current values for position, color, and font information.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPQuery(
    TBHANDLE      in_pageHandle,
    long*         outp_xCoordinate,
    long*         outp_yCoordinate,
    MODCAColors*  outp_color,
    fontID*       outp_fontID,
    long*         outp_fontlinesp
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPQUERY.
    CALL "CBLQuery"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
            BY REFERENCE
                AFP-X-COORDINATE
                AFP-Y-COORDINATE
                AFP-COLOR
                AFP-FONT-ID
                AFP-FONT-LINESPACING
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLQuery" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPQUERY      PR          10I 0  EXTPROC('AFPQuery')
D  AFPPAGHDL          *    VALUE
D  AFPXCOORD          10I 0
D  AFPYCOORD          10I 0
D  AFPCOLOR           10I 0
D  AFPFONTID           5I 0
D  AFPFONTLS          10I 0
```

### Input Parameters

#### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Output Parameters

#### x coordinate

The current x (inline) position. This is the last position set with the Set Position, Horizontal Move To, Put Text, or Horizontal Move calls. The coordinate will be returned in the current units if a Set Units call was issued (COBOL only).

## Query

### y coordinate

The current y (baseline) position. This is the last position set with the Set Position, Next Line, Vertical Move To, Put Text, or Vertical Move calls. The coordinate will be returned in the current units if a Set Units call was issued (COBOL only).

### Color

The current color from the Set Color command.

### Font ID

The ID of the current font from the Set Font command.

### Font line spacing

Amount of vertical space from one line of text in the paragraph to the next in the font currently being used. Vertical space is measured from the baseline of text to the next baseline.

## Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

Find out the current state of things:

- **For the AFP Toolbox C Library:**

```
AFPQuery(PageHandle, &Xpos, &Ypos, &Color,  
         &FontID, &Fontlinesp);
```

- **For the AFP Toolbox COBOL Interface:**

```
* Do an AFPQuery  
PERFORM AFPQUERY.  
    DISPLAY AFP-X-COORDINATE.  
    DISPLAY AFP-Y-COORDINATE.  
    DISPLAY AFP-COLOR.  
    DISPLAY AFP-FONT-ID.  
    DISPLAY AFP-FONT-LINESPACING.
```

- **For the AFP Toolbox RPG Interface:**

```
C                EXSR        AFPQUERYR
```

## Repeat String

### Function

Repeat a string of characters at the current position. The text must be ended before you end the page. Ensure that the text you have specified fits on the logical page.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPRepeatStr(
    TBHANDLE    in_pageHandle,
    short        in_stringLength,
    char*        in_repeatedString
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPRSTR.
    CALL "CBLRepeatStr"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-REPEAT-LENGTH
            BY CONTENT
                AFP-REPEAT-STRING
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLRepeatStr" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPRPTSTR      PR          10I 0 EXTPROC('AFPRepeatStr')
D  AFPPAGHDL          *  VALUE
D  AFPRPTLEN          5I 0  VALUE
D  AFPRPSTRN          250
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### String length

Number of total characters to produce in the text block.

### Repeated string

A string of characters to repeat. The string is repeated as many times as necessary to equal string-length. If the length is not an integral multiple of the string of characters, the last repetition of the string is truncated. If the length is less than the number of characters in the string, then the string is truncated.

## Return Codes

- 4      Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Repeat String

### Sample Function Call

- **For the AFP Toolbox C Library:**

To create a string of 100 dots and 100 blanks:

```
strcpy(TextString, ". ");  
rc = AFPRepeatStr(PageHandle, 200, TextString);
```

- **For the AFP Toolbox COBOL Interface:**

This example moves the current position to one inch across, one inch down, and repeats the period/blank character sequence 100 times.

\* SetPos

```
MOVE 1440 TO AFP-X-COORDINATE.  
MOVE 1440 TO AFP-Y-COORDINATE.  
MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.  
MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
```

PERFORM AFPSPOS.

\* The RepeatString xmp

```
MOVE ". " TO AFP-REPEAT-STRING.  
MOVE 200 TO AFP-REPEAT-LENGTH.  
MOVE 2 TO AFP-STRING-LENGTH.
```

PERFORM AFPRSTR.

- **For the AFP Toolbox RPG Interface:**

This example moves the current position to one inch, one inch and repeats the period/blank character sequence 100 times.

```
C      MOVEL      ". "      AFPRSTRN  
C      MOVEL      200      AFPRSTLEN  
C      EXSR      AFPRPTSTR
```

## Rotate Overlay

### Function

Includes a page overlay at the current location and gives a rotation value.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPRotateOvly(
    TBHANDLE    in_pageHandle,
    DegRot      in_rotation,
    char*        in_overlay_name
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPROVL.
    CALL "CBLRotateOvly"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-OBJECT-ROTATION
            BY CONTENT
                AFP-OVLY-NAME
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLRotateOvly" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

#### Rotation

The rotation for the overlay. The valid values are:

**DEG\_0 (for C), DEG-0 (for COBOL), or DEG0 (for RPG)**

The overlay is not rotated.

**DEG\_90 (for C), DEG-90 (for COBOL), or DEG90 (for RPG)**

The overlay is rotated 90 degrees clockwise.

**DEG\_180 (for C), DEG-180 (for COBOL), or DEG180 (for RPG)**

The overlay is rotated 180 degrees clockwise.

**DEG\_270 (for C), DEG-270 (for COBOL), or DEG270 (for RPG)**

The overlay is rotated 270 degrees clockwise.

#### Overlay name

The eight character name of the overlay member in the PSF resource library.

### Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

### Sample Function Call

Include an overlay named MYOVLY and rotate it 180 degrees:

- **For the AFP Toolbox C Library:**

## Rotate Overlay

```
rc = AFPRotateOverlay (PageHandle,"MYOVLY",DEG_180) ;
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE "OVL180" TO MYOVLY.  
MOVE DEG-180 TO AFP-OBJECT-ROTATION.  
PERFORM AFPROVL.
```

## Set Color

### Function

Specifies the color for subsequent data (text and rules).

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPSetColor(
    TBHANDLE      in_pageHandle,
    MODCAColors   in_color
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPSCLR.
    CALL "CBLSetColor"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-COLOR
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLSetColor" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPSETCOL      PR          10I 0 EXTPROC('AFPSetColor')
D AFPPAGHDL      *          *  VALUE
D AFPCOLOR      10I 0  VALUE
```

### Input Parameters

#### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

#### Color

The color to print. Valid values are:

- BLUE, RED, MAGENTA, PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, GRAY, or NONE (for C, COBOL, or RPG)
- MEDIUM\_COLOR (for C) , MEDIUM-COLOR (for COBOL), or MEDIUMCOL (for RPG)
- DEFAULT\_COLOR (for C), DEFAULT-COLOR (for COBOL), or DEFAULTCOL (for RPG)

### Return Codes

- 4** Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 10** You specified an unsupported color. Select a color from the list above.

### Sample Function Call

- **For the AFP Toolbox C Library:**

To produce subsequent text in blue:

```
rc = AFPSetColor(PageHandle, BLUE);
```

- **For the AFP Toolbox COBOL Interface:**

## Set Color

This example produces a six inch yellow vertical rule at the current position.

```
* SetColor
      MOVE YELLOW      TO AFP-COLOR.
      PERFORM AFPSCLR.
* PutVRule 6 inch rule
      MOVE 8640  TO AFP-RULE-LENGTH.
      MOVE 360   TO AFP-RULE-THICKNESS.
      PERFORM AFPPVRUL.
```

- **For the AFP Toolbox RPG Interface:**

To produce subsequent text in blue:

```
C          Z-ADD      BLUE      AFPCOLOR
C          EXSR       AFPCOLORR
```



## Set Data Stream Codepage

### Function

Specifies the code page name used for encoding character strings, such as names, in the output data stream. For example, the FQN (fully-qualified name) triplet on the BDT (Begin Document) structured field uses this code page for its encoding. The code page must exist in the font library.

If this function is not issued, code page T1V10500 is assumed for all character strings in the output data stream.

This function does not control the code page used for encoding text strings in the output data stream; text encoding is controlled by the code page specified when the font is defined and by the Set Input Codepage and Set Text Codepage functions.

This call must be issued before the Begin Document call for it to affect that document's datastream.

**Note:** Do not use this function unless you are completely familiar with code pages.

### Syntax

#### For the AFP Toolbox C Library:

```
AFPSetDSCodepage(
    TBHANDLE    in_sessHandle,
    char*        in_codepage,
    char         in_spacechar
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPSDSCP.
    CALL "CBLSetDSCodepage"
        USING
            BY VALUE
                AFPAPI-HANDLE
            BY CONTENT
                AFP-CODE-PAGE
                AFP-SPACE-CHAR
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLSetDSCodepage" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

#### For the AFP Toolbox RPG Interface:

```
DAFPSETTXCP      PR          10I 0  EXTPROC('AFPSetTextCodepage')
D  AFPAPIHDL          *  VALUE
D  AFPTXCPAG          9
D  AFPSPCHAR          2
```

### Input Parameters

#### Session handle

The handle for the session. Session handle is returned by the Begin Session call.

#### Codepage name

Up to eight characters to use as the code page for encoding the character strings in output data stream. If more than eight are specified, only the first eight characters are used.

## Set Data Stream Codepage

### Space character

Specify a single byte character that is the code point for a blank in the specified code page. This is normally X'20' for ASCII code pages and X'40' for EBCDIC.

## Return Codes

- 2      Session handle is null or is not valid. Make sure you have called Begin Session before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

This example sets the code page for the output data stream to be 850:

```
char output="T1000850";
AFPSetDSCodepage (sesshandle,outputcp,' ');
```

- **For the AFP Toolbox COBOL Interface:**

This example sets the output code page to T1000850.

```
PERFORM AFPINIT.
* Ascii capital A B C in CP 850 are hex  41, 42, 43
* Ascii capital D E F in CP 850 are hex  44, 45, 46
* Ascii capital G H I in CP 850 are hex  47, 48, 49
* SetDSCodepage Example
  MOVE "T1000850" TO AFP-CODE-PAGE.
```

- **For the AFP Toolbox RPG Interface:**

This example sets the output code page to T1000850.

C	MOVEL	"T1000850"	AFPTXCPAG
C	MOVEL	" "	AFPSPCHAR
C	EXSR	AFPSDCPR	

## Set Extended Color

### Function

The Set Extended Color control sequence specifies a color value and defines the color specified encoding for that value. The specified color value is applied to the foreground areas of the presentation space (i.e., text strings within PText objects).

### Syntax

**For that AFP Toolbox C Interface:**

```
int AFPSetExtendedColor(
    TBHANDLE    in_pageHandle,
    ColorSpace in_colortype,
    MODCAColors in_color,
    uint in_colspec1,
    uint in_colspec2,
    uint in_colspec3,
    uint in_colspec4,
);
```

**For that AFP Toolbox COBOL Interface:**

```
AFPSECL.
    CALL "CBLSetExtendedColor"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-COLOR-TYPE
                AFP-COLOR
                AFP-COLSPEC1
                AFP-COLSPEC2
                AFP-COLSPEC3
                AFP-COLSPEC4
        RETURNING
            AFP-RET-CODE.
```

### Input Parameters

#### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

#### Color Space Type

Specifies the type of color to generate:

##### **RGB\_COLOR (for C) or RGB-COLOR (for COBOL)**

The color value is specified as three intensity levels for the red, green, and blue components of the color. These numbers are given for color specification values 1, 2, and 3. The fourth specification is ignored and should be specified as 0.

##### **CMYK\_COLOR (for C) or CMYK-COLOR (for COBOL)**

The color value is specified as four intensity levels for the cyan, magenta, yellow, and black components of the color. These are color specifications 1-4.

##### **HILIGHT\_COLOR (for C) or HILIGHT-COLOR (for COBOL)**

Defines a request for presentation device to generate a highlight color. Specify a MO:DCA Color value for the highlight color. Color specification 1 is the percentage coverage for the specified color, and color specification 2 is the percentage of black that is added to the coverage (can be 0). Color specifications 3 and 4 should be 0.

## Set Extended Color

### **CIELAB\_COLOR (for C) or CIELAB-COLOR (for COBOL)**

Specified with three components. Color specification 1 is the luminance, and color specifications 2 and 3 are the chrominance differences. Color specification 4 should be 0.

### **OCA\_COLOR (for C) or OCA-COLOR (for COBOL)**

Defines a request for presentation device to generate a highlight color. Specify a MO:DCA Color value for the highlight color. The presentation device will produce printed colors using a color mapping table (see PSF documentation for your environment). Color specifications 1-4 should be 0.

### **MODCAColors in\_color**

The color to print.

### **Color specifications 1-4**

Usage depends on the color space type, see descriptions above.

## Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

### • **For the AFP Toolbox C Interface:**

To produce a CMYK color space using an intensity range of 50% for cyan, and 0% for magenta, yellow and black.

```
rc = AFPSetExtendedColor(PageHandle, CMYK_COLOR, YELLOW,50,0,0,0);
```

To produce a CMYK color space using an intensity range of 50% for cyan, magenta, yellow and black.

```
rc = AFPSetExtendedColor(PageHandle, CMYK_COLOR, YELLOW, 50,50,50,50);
```

To produce a RGB color space using binary numbers that specify the red, green, and blue intensity values.

```
rc = AFPSetExtendedColor(PageHandle, RGB_COLOR, RED,255,128,1,0);
```

To produce a CIELAB color space using a luminance difference of 75, and chrominance differences of 10 and 20.

**Note:** The luminance range is 0 - 100. The chrominance ranges are -127 to 127.

```
rc = AFPSetExtendedColor(PageHandle, CIELAB_COLOR, NULL,75,10,20,0);
```

To produce an OCA color space using blue for color.

**Note:** The luminance range is 0 - 100. The chrominance ranges are -127 to 127.

```
rc = AFPSetExtendedColor(PageHandle, OCA_COLOR, NULL,NULL,NULL,NULL);
```

### • **For the AFP Toolbox COBOL Interface:**

```
SET AFP-CURRENT-HANDLE TO AFP-PAGE-HANDLE  
MOVE RGB-COLOR TO AFP-COLOR-TYPE  
MOVE RED TO AFP-COLOR  
MOVE 10 TO AFP-COLORSPEC1  
MOVE 20 TO AFP-COLORSPEC2  
MOVE 30 TO AFP-COLORSPEC3  
MOVE 0 TO AFP-COLORSPEC4  
PERFORM AFPSECL.
```

**Note:** Color options are red, magenta, green, cyan, yellow, black, and brown.

## Set Font

### Function

Specifies the font for subsequent text data. Do not issue this call within a paragraph or a table.

### Syntax

**For that AFP Toolbox C Library:**

```
int AFPSetFont(
    TBHANDLE    in_pageHandle,
    FontID      in_font_id
);
```

**For that AFP Toolbox COBOL Interface:**

```
AFPSFONT.
    CALL "CBLSetFont"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-FONT-ID
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLSetFont" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For that AFP Toolbox RPG Interface:**

```
DAFPSFONT      PR          10I 0 EXTPROC('AFPSetFont')
D AFPPAGHDL      *          * VALUE
D AFPFNTID      5I 0 VALUE
```

### Input Parameters

#### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

#### Font ID

The font ID returned by the Define Font By Name, Define Font By Attribute calls, or other calls that define fonts.

### Return Codes

- 4** Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 8** Font not defined. You must successfully define the font with Define Font By Name or Define Font By Attribute, or other calls that define fonts before issuing Set Font.
- 14** This function is not allowed within a paragraph. Issue End Paragraph before issuing this call.

### Sample Function Call

- **For the AFP Toolbox C Library:**

Define and start a 12 point Times font:

```
pt12 = 120;
AFPDefineFontAttr(currentdoc, cp500, "TIMES NEW ROMAN LATIN1",
    pt12, MEDIUM_WT, MEDIUM_WD, NORMAL_FS, &tnr12pt);
rc = AFPSetFont(PageHandle, tnr12pt);
```

## Set Font

- **For the AFP Toolbox COBOL Interface:**

This example defines and starts 14 point Times New Roman with code page T1V10500 and writes a string.

```
* DefineFontByAttr                                     *
    MOVE "T1V10500" TO AFP-CODE-PAGE.
    MOVE "TIMES NEW ROMAN LATIN1" TO AFP-DESCRIPTIVE-NAME.
    MOVE 140 TO AFP-POINT-SIZE.
    MOVE MEDIUM TO AFP-WEIGHT.
    MOVE NORML TO AFP-FONT-WIDTH.
    MOVE ROMAN TO AFP-STYLE.
    PERFORM AFPDFNTAT.
    MOVE AFP-FONT-ID TO TIM14TYPE.
* SetFont
    MOVE TIM14TYPE TO AFP-FONT-ID.
    PERFORM AFPSFONT.
* AFPWriteString
    MOVE "HELLO WORLD" TO AFP-CHARACTER-STRING.
    MOVE 11 TO AFP-STRING-LENGTH.
    MOVE LEFT-ALIGN TO AFP-ALIGNMENT-OPTION.
    MOVE FALS TO AFP-POSITION-OPTION.
    MOVE -1 TO AFP-EXTRA-VARSPACE.
    MOVE -1 TO AFP-EXTRA-ICSPACE.
    MOVE "." TO AFP-ALIGNMENT-CHAR.
    PERFORM AFPWRITE.
```

- **For the AFP Toolbox RPG Interface:**

\*Find a font that matches the specified attributes and return a font ID.

C	DTA(6)	CAT(P)	X'00:0	AFPDENAM
C	DTA(7)	CAT(P)	X'00:0	AFPCDEPAG
C		Z-ADD	140	AFPPNTSIZ
C		Z-ADD	MEDIUM	AFPWGT
C		Z-ADD	NORMAL	AFPFNTWTH
C		Z-ADD	ROMAN	AFPSTL
C		EXSR	AFPDFNTATR	
C		Z-ADD	AFPFNTID	TIM14TYPE
*Specify font for subsequent text data				
C		Z-ADD	TIM14TYPE	AFPFNTID
C		EXSR	AFPSFONTR	

## Set Font Type

### Function

Specifies the font type for subsequent text data.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPSetFontType(
    FontCSType in_type
);
```

**For the AFP Toolbox COBOL Interface:**

AFPSFTYP.

```
CALL "CBLSetFontType"
  USING
  BY VALUE
  AFP-FONT-TYPE
  RETURNING
  AFP-RET-CODE.
MOVE "CBLSetFontType" TO AFP-ERRDATA.
PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPSFTYPE      PR          10I 0 EXTPROC('AFPSetFontType')
D AFPFTYPE      10I 0 VALUE
```

## Input Parameters

### Type

The font type can be

**FONT\_240, FONT\_300, or FONT\_0LN (for C)**

**FONT-240, FONT-300, or FONT-0LN (for COBOL)**

**FONT240, FONT300, or FONT0LN (for RPG).**

It describes the font technology used for all font definitions in the document. The font type remains active until it is changed with another Set Font Type call.

### OS/2® and AIX Systems

The font type defines the file extension for the font members (for example, .240, .300, or .0ln in the font name).

### MVS and OS/400 Systems

The font type is used to determine which AFP font technology to use. The AFP Toolbox searches all of the font libraries specified with MVS FNTLIBDD or in the OS/400 library list and uses the first member found that matches all of the attributes, including font technology. However, since FONT\_240 and FONT\_300 libraries have the same member name, the AFP Toolbox cannot distinguish between these and will read the first one in the search order.

The default in all environments is FONT\_240.

## **Set Font Type**

## **Return Codes**

- 4      Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## **Sample Function Call**

For an example of setting the font type, see “Define Font By Attributes With Scaling” on page 123.



## Set Input Codepage

### Function

Specifies the code page name that is used for encoding all input character strings, such as names and text, that are parameters on the AFP Toolbox functions. This code page is used for all input character strings unless it is changed with another Set Input Codepage request or temporarily overridden (for text only) within a page by Set Text Codepage.

The default input code page for all text in AFP Toolbox is multinational ASCII code page T1000850 for workstation systems and EBCDIC T1V10500 for host systems.

This code page does not control the code page used in an open page, or for encoding text strings in the output data stream. Text encoding is controlled by the code page specified when the font is defined. Text is automatically translated from the code page specified with Set Input Codepage to the code page associated with the font when the text is placed in the output data stream.

### Syntax

#### For the AFP Toolbox C Library:

```
AFPSetInputCodepage(
    TBHANDLE    in_sessHandle,
    char*        in_codepage,
    char         in_spacechar
);
```

#### For the AFP Toolbox COBOL Interface:

```
AFPSINCP.
    CALL "CBLSetInputCodepage"
        USING
            BY VALUE
                AFPAPI-HANDLE
            BY CONTENT
                AFP-CODE-PAGE
                AFP-SPACE-CHAR
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLSetInputCP" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

#### For the AFP Toolbox RPG Interface:

```
DAFPSETINCP      PR          10I 0  EXTPROC('AFPSetInputCodepage')
D  AFPAPIHDL          *  VALUE
D  AFPINCPAG          9
D  AFPSPCHAR          2
```

## Input Parameters

### Session handle

The handle for the session. Session handle is returned by the Begin Session call.

### Codepage name

Up to eight characters to use as the code page for encoding the character strings in output data stream. If more than eight are specified, only the first eight characters are used.

### Space character

Specify a single byte character that is the code point for a blank in the specified code page. This is normally X'20' for ASCII code pages and X'40' for EBCDIC.

## Set Input Codepage

## Return Codes

2 Session handle is null or invalid. Make sure you have called Begin Session before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

This example shows how to print a solid triangle (▲) using codepage T1000437 as input and T1000363 codepage as output.

```
char inputcp = "T1000437";
char outputcp = "T1000363";
char uptriangle[] = "\x1E";
FontID symfont;
```

```
AFPSetInputCodepage(sesshandle,inputcp,' ');
...
AFPDefineFontByAttr(outputcp,"SYMBOLS",90,MEDIUM_WT,
                    MEDIUM_WD,NORMAL_FS,&symfont);
AFPSetFont(currentpage,symfont);
AFPWriteString(currentpage,uptriangle,-1,-1,LEFT_ALIGN,' ',TRUE);
```

- **For the AFP Toolbox COBOL Interface:**

This example sets T1GI0395 as the input code page. The output code page is T1V10500. This example shows how X'4A' from code page T1GI0395 produces X'4F' in the output.

```
PERFORM AFPINIT.
*-----
* 4A in codepage 395 is the exclamation point.
* In codepage 500 the exclamation point is at codepoint 4F.
* Data item HEX-4A is codepage 4A
* This should produce the hex 4F in the output
*-----
* SetInputCodepage Example
  MOVE "T1GI0395" TO AFP-CODE-PAGE.
  MOVE " " TO AFP-SPACE-CHAR.
PERFORM AFPSINCP.
*BeginDoc
  MOVE FILED TO AFP-OUTPUT-TYPE.
  MOVE "MYOUTPT"
    TO AFP-OUTPUT-FILENAME.
PERFORM AFPBDOC.
* BgnPage 1
PERFORM AFPBPAGE.
*
  MOVE "T1V10500" TO AFP-CODE-PAGE.
  MOVE "TIMES NEW ROMAN LATIN1"
    TO AFP-DESCRIPTIVE-NAME.
  MOVE 90 TO AFP-POINT-SIZE.
  MOVE MEDIUM TO AFP-WEIGHT.
  MOVE NORML TO AFP-FONT-WIDTH.
  MOVE ROMAN TO AFP-STYLE.
PERFORM AFPDFNTAT.
  MOVE AFP-FONT-ID TO TIM09TYPE.
  MOVE TIM09TYPE TO AFP-FONT-ID.
PERFORM AFPSFONT.
* SetPos
  MOVE 1440 TO AFP-X-COORDINATE.
  MOVE 1440 TO AFP-Y-COORDINATE.
  MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
  MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
PERFORM AFPSPOS.
* Put text on the page
  MOVE HEX-4A
    TO AFP-CHARACTER-STRING.
  MOVE 1 TO AFP-STRING-LENGTH.
  MOVE LEFT-ALIGN TO AFP-ALIGNMENT-OPTION.
```

```

        MOVE FALS TO AFP-POSITION-OPTION.
        MOVE -1 TO AFP-EXTRA-VARSPACE.
        MOVE -1 TO AFP-EXTRA-ICSPACE.
        MOVE "." TO AFP-ALIGNMENT-CHAR.
    PERFORM AFPWRITE.
* EndPage
    PERFORM AFPEPAGE.
* EndDoc
    PERFORM AFPEDOC.
* EndSession
    PERFORM AFPEND.

```

- **For the AFP Toolbox RPG Interface:**

This example sets T1GI0395 as the input code page. The output code page is T1V10500. This example shows how X'4A' from code page T1GI0395 produces X'4F' in the output.

```

C          MOVEL      'T1GI0395'    AFPINCPAG
C          MOVE       ' '           AFPSPCHAR
C          EXSR       AFPSICPR
C      DTA(6)  CAT(P)   x'00':0      AFPDESNAM
C      DTA(7)  CAT(P)   x'00':0      AFPCDEPAG
C          Z-ADD      140           AFPPNTSIZ
C          Z-ADD      MEDIUM       AFPWGT
C          Z-ADD      NORMAL        AFPFNTWTH
C          Z-ADD      ROMAN         AFPSTL
C          EXSR       AFPDFNTATR
C          EXSR       AFPSFONTR
C          MOVE       X'4A'         AFPCHRSTR
C          MOVE       CENTRALGN     AFPALNOPT
C          MOVE       FALS          AFPPOSOPT
C          Z-ADD      -1            AFPXTRVAR
C          Z-ADD      -1            AFPXTRICS
C          MOVE       '.'           AFPALNCHR
C          EXSR       AFPWRITER
...
TIMES NEW ROMAN LATIN1             6
T1V10500                           7

```

# Set Input Double-Byte Codeset

## Function

Specifies the code set name that is used for encoding double-byte text strings. Unlike Set Input Codepage, this call has no effect on names such as the FQN triplet on the BDT.

Text is automatically translated from the codeset specified with the Set Input Double-byte Codeset call to the code set associated with the font when the DBCS text is placed in the output data stream.

## Syntax

### For the AFP Toolbox C Library:

```
AFPSetInputDbCodeset(  
    TBHANDLE    in_sessHandle,  
    char*        in_codeset,  
    char         in_spacechar  
);
```

### For the AFP Toolbox COBOL Library:

```
AFPSIDBCS.  
    CALL "CBLSetInputDbCodeset"  
        USING  
        BY CONTENT  
        AFP-CODE-SET  
        AFP-SPACE-CHAR  
        RETURNING  
        AFP-RET-CODE.  
    MOVE "CBLSetInputDbCS" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

## Input Parameters

### Session handle

The handle for the session. Session handle is returned by the Begin Session call.

### Code set name

The code set name for encoding the character strings in the input character string.

### Space character

Specify a single byte character that is the code point for a blank in the specified code page. This is normally X'20' for ASCII code pages and X'40' for EBCDIC.

## Return Codes

- 2      Session handle is null or invalid. Make sure you have called Begin Session before issuing this call.

## Sample Function Call

Set the code set for the text character in the session:

- **For the AFP Toolbox C Library:**

```
TBHANDLE toolboxSession;  
  
AFPSetInputDbCodeset(  
    toolboxSession, "IBM-932", ' '  
);
```

- **For the AFP Toolbox COBOL Interface:**

## Set Input Double-Byte Codeset

```
MOVE "IBM-932" TO AFP-CODE-SET.  
MOVE SPACES to AFP-SPACE-CHAR.  
PERFORM AFPSIDBCS.
```

---

# Set Media Size

## Function

Set the size of the media (printable area) for subsequent pages in the document. If this call is not specified, the default of 8.5 x 11 inches is used for all pages.

**Note:** Use Set Media Size for actual physical paper sizes other than 8.5 x 11 inches. To rotate the logical page, that is, to specify portrait or landscape orientation, use Set Text Orientation and **do not** issue Set Media Size.

## Syntax

**For the AFP Toolbox C Library:**

```
int AFPSetMediaSize(
    TBHANDLE    in_docHandle,
    long        in_pageWidth,
    long        in_pageLength
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPSMSIZ.
    CALL "CBLSetMediaSize"
        USING
            BY VALUE
                AFP-DOCUMENT-HANDLE
                AFP-DOC-PAGE-WIDTH
                AFP-DOC-PAGE-DEPTH
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLSetMediaSize" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPSETMSZ      PR          10I 0 EXTPROC('AFPSetMediaSize')
D  AFPDOCHDL          *  VALUE
D  AFPPAGWID          10I 0  VALUE
```

## Input Parameters

### Document handle

The handle for the current document. Document handle is returned by the Begin Document call.

### Page width

The width of the page. The maximum value for page width is 22 inches in 1440ths of an inch.

### Page length

The length of the page. The maximum value for page length is 22 inches in 1440ths of an inch.

## Return Codes

- 3      The document handle is not pointing to a valid document. Make sure you have called Begin Document before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Set the page size to 8.5 by 14 inches. (Issue this call after the AFBgnDoc all, but before the first AFBgnPage call.)

```
width = IN_2_U1440(8.5);
length = IN_2_U1440(14);
rc = AFPSetMediaSize(DocHandle, width, length);
```

- **For the AFP Toolbox COBOL Interface:**

This example sets the page size to 8.5 inches by 14 inches. Issue the Set Media Size call after the Begin Document call and before the first Begin Page call.

```
PERFORM AFBDOC.
* 8.5x14 page size
    MOVE 12240 TO AFB-DOC-PAGE-WIDTH.
    MOVE 20160 TO AFB-DOC-PAGE-DEPTH.
PERFORM AFBMSIZ.
PERFORM AFBPAGE.
```

- **For the AFP Toolbox RPG Interface:**

This example sets the page size to 8.5 inches by 14 inches. Issue the Set Media Size call after the Begin Document call and before the first Begin Page call.

```
C          Z-ADD      12240      AFPPAGPWID
C          Z-ADD      20160      AFPPAGDEP
C          EXSR        AFPSMSZR
```

# Set Object Color Profile

## Function

Identifies the color profile to use when printing the specified object on an Infoprint Color 130 Plus printer. This ensures the color integrity of the object when it is printed.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPSetObjectColorProfile(
    AString&      in_objectName,
    ObjectType     in_objectType ,
    OtherObjectType in_otherObjType ,
    ColorProfile   in_colorProfile
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPSOCP.
CALL "CBLSetObjCP"
    USING
    BY VALUE
        AFP-PAGE-HANDLE
    BY CONTENT
        AFP-OBJECT-NAME
    BY VALUE
        AFP-OBJECT-TYPE
        AFP-OTHER-OBJECT-TYPE
        AFP-COLOR-PROFILE
    RETURNING
        AFP-RET-CODE.
MOVE "CBLSetObjCP" TO AFP-ERRDATA.
MOVE SPACES TO AFP-OBJECT-NAME(AFP-NAME-LENGTH:1)
PERFORM CHKSUCC.
```

## Input Parameters

### Object name

The name of the object to be printed with the specified color profile.

### Object type

The type of object that is to be printed with the specified color profile. Values are:

#### PAGE\_SEGMENT\_OT (for C) or PAGE-SEGMENT-OT (for COBOL)

A MO:DCA page segment resource object containing an IOCA image.

#### GRAPHICS\_OT (for C), GRAPHICS-OT for COBOL, or GRAPHICOT (for RPG)

A graphics object with MO:DCA syntax as defined in *Graphics Object Content Architecture Reference*

#### IMAGE\_OT (for C), IMAGE-OT (for COBOL), or IMAGEOT (for RPG)

An image object with MO:DCA syntax as defined in *Image Object Content Architecture Reference*

#### BAR\_CODE\_OT (for C), BAR-CODE-OT (for COBOL), or BARCODEOT (for RPG)

A bar code object with MO:DCA syntax as defined in *Bar Code Object Content Architecture Reference*

#### OTHER\_OBJECT\_DATA\_OT (for C and C++), OTHER-OBJECT-DATA-OT (for COBOL), or OTHEROT (for RPG)

Object data to include that is not defined by an IBM presentation architecture.

### Other object type

Use this parameter to fully identify the object type when Object Type is set to Other. Values are:

Value	Description
-------	-------------



## Set Object Color Profile

EPS_00T (for C), EPS-00T (for COBOL)	EPS
TIFF_00T (for C), TIFF-00T (for COBOL)	TIFF
DIB_WIN_00T (for C), DIB-WIN-00T (for COBOL)	DIB, Windows version
DIB_OS2_00T (for C), DIB-OS2-00T (for COBOL)	DIB, OS/2 PM version
PCX_00T (for C), PCX-00T (for COBOL)	PCX
GIF_00T (for C), GIF-00T (for COBOL)	GIF
JFIF_00T (for C), JFIF-00T (for COBOL)	JFIF
PDF_SP0_00T (for C), PDF-SP0-00T (for COBOL)	PDF Single Page Object
PCL_PO_00T (for C), PCL-PO-00T (for COBOL)	PCL Page Object
EPS_TRAN_00T (for C), EPS-TRAN-00T (for COBOL)	EPS with Transparency
PDF_TRAN_00T (for C), PDF-TRAN-00T (for COBOL)	PDF with Transparency

### Color profile

The color profile to use when printing the specified object. Values are:

Value	Description
CMYK_SWOP_CP (for C) CMYK-SWOP-CP (for COBOL)	US CMYK SWOP standard
CMYK_EURO_CP (for C) CMYK-EURO-CP (for COBOL)	European CMYK Euroscale standard

### Return Codes

- 23** The object type is not valid. See the Object type parameter for valid values.
- 24** Object type is other, but the other object type is not valid. See the Other object type parameter for valid values.
- 25** The color profile is not valid. See the Color profile parameter for valid values.

---

# Set Position

## Function

Sets the new position for output on the page. The initial position on the page is at (0,0). Do not issue this call within a paragraph or a table.

## Syntax

**For the AFP Toolbox C Library:**

```
int AFPSetPos(
    TBHANDLE    in_pageHandle,
    short        in_inlinePosition,
    POSTYPE      in_inlineReference,
    short        in_baselinePosition,
    POSTYPE      in_baselineReference
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPSPOS.
    CALL "CBLSetPos"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-X-COORDINATE
                AFP-X-REF-COORD-SYS
                AFP-Y-COORDINATE
                AFP-Y-REF-COORD-SYS
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLSetPos" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPSPOS          PR          10I 0 EXTPROC('AFPSetPos')
D AFPPAGHDL        *          VALUE
D AFPXCRD          10I 0 VALUE
D AFPXRFCRDS       10I 0 VALUE
D AFPYCRD          10I 0 VALUE
D AFPYRFCRDS       10I 0 VALUE
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Inline position

New horizontal position (x). Negative values are not valid if the inline reference is set to ABSOLUTE\_POS.

### Inline reference

Specifies whether x is absolute from the current object origin or relative to the last x position. The valid values are:

```
RELATIVE_POS
ABSOLUTE_POS
```

### Baseline position

New vertical position (Y). Negative values are not valid if the baseline reference is set to ABSOLUTE\_POS.

**Baseline reference**

Specifies whether Y is absolute from the current object origin or relative to the last Y position. The valid values are:

RELATIVE\_POS  
ABSOLUTE\_POS

**Return Codes**

- 4** Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 10** you have specified a reference coordinate other than ABSOLUTE\_POS or RELATIVE\_POS.
- 14** This function is not allowed within a paragraph. Issue End Paragraph before issuing this call.

**Sample Function Call**

- **For the AFP Toolbox C Library:**

Set the new position to be exactly two inches in and three inches down on the logical page:

```
hpos = IN_2_U1440(2);
vpos = IN_2_U1440(3);
rc = SetPosition(PageHandle, hpos, ABSOLUTE_POS, vpos, ABSOLUTE_POS);
```

- **For the AFP Toolbox C Library:**

This command sets the position for the table at 20 millimeters in the X-direction and 80 millimeters in the Y direction.

```
AFPSetPos(hPage, MM_2_U1440(20), ABSOLUTE_POS,
          MM_2_U1440(80),ABSOLUTE_POS);
```

- **For the AFP Toolbox COBOL Interface:**

This example establishes the current position at two inches across the page and three inches down on the logical page.

```
MOVE 2880 TO AFP-X-COORDINATE.
MOVE 4320 TO AFP-Y-COORDINATE.
MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
PERFORM AFPSPOS.
```

- **For the AFP Toolbox RPG Interface:**

This example establishes the current position at two inches across the page and three inches down on the logical page.

C	Z-ADD	2880	AFPXCRD
C	Z-ADD	4320	AFPYCRD
C	Z-ADD	ABSPPOS	AFPXRFCRDS
C	Z-ADD	ABSPPOS	AFPYRFCRDS
C	EXSR	AFPSPOS	

---

# Set Text Codepage

## Function

Specifies the code page name that is used for encoding input text character strings, such as the text provided as input to the Put Text, Repeat String, and Write String functions. With the exception of the Measure String function call, this code page is used for all text strings within this page until it is changed with another Set Text Codepage request. The code page must exist in the font library.

This function is used to temporarily override the code page requested with Set Input Codepage (or its default) for the text strings on this page only. See “Set Input Codepage” on page 225 for more information.

Text is automatically translated from the code page specified with Set Text Codepage to the output code page associated with the current font when the text is placed in the output data stream.

## Syntax

### For the AFP Toolbox C Library:

```
AFPSetTextCodepage(  
    TBHANDLE    in_pageHandle,  
    char*       in_codepage,  
    char*       in_spacechar  
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPSTXCP.  
    CALL "CBLSetTextCodepage"  
        USING  
            BY VALUE  
                AFP-PAGE-HANDLE  
            BY CONTENT  
                AFP-CODE-PAGE  
                AFP-SPACE-CHAR  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLSetTextCP" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPSETTXCP      PR              10I 0  EXTPROC('AFPSetTextCodepage')  
D  AFPPAGHDL      *              VALUE  
D  AFPTXCPAG      9  
D  AFPSPCHAR      2
```

## Input Parameters

### Page handle

The handle for the page. Page handle is returned by the Begin Page call.

### Codepage name

Up to eight characters to use as the code page for encoding the character strings in output data stream. If more than eight are specified, only the first eight characters are used.

### Space character

Specify a single byte character that is the code point for a blank in the specified code page. This is normally X'20' for ASCII code pages and X'40' for EBCDIC.

## Return Codes

- 4 Page handle is null or is not valid. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

This sample demonstrates temporarily setting the text code page to print some Greek symbols. It converts the greek characters from code page T1000850 (codepoints X'D1', X'D2', and X'D3') to output code page T1000875 (codepoints X'63', X'64', and X'65'). This conversion is done automatically by the Toolbox during the AFP Write String processing.

```
char cp500[] = "T1V10500"; /* English output codepage */
char cp875[] = "T1000875"; /* Greek output codepage */
char cp850[] = "T1000850"; /* English input codepage */
char cp851[] = "T1000851"; /* Greek input codepage */
char txtstr[] = "Some English Text";
char greekchars[] = "\xD1\xD2\xD3";

rc = AFPDefineFontByAttr(cp500,"TIMES NEW ROMAN LATIN1",
    pt12,MEDIUM_WT,MEDIUM_WD,NORMAL_FS,&f1);
rc = AFPDefineFontByAttr(cp875,"BOLDFACE GREEK",
    pt12,SEMIBOLD_WT,MEDIUM_WD,ITALIC_FS,&f2);
AFPSetFont(currentpage,f1);

AFPSetPos(currentpage,1440,ABSOLUTE_POS, 1440, ABSOLUTE_POS);
AFPWriteString(currentpage,txtstr,-1,-1,LEFT_ALIGN,' ',FALSE);

AFPSetTextCodepage(currentpage,cp851,' ');
AFPSetFont(currentpage,f2);
AFPNextLine(currentpage);
AFPWriteString(currentpage,greekchars,-1,-1,LEFT_ALIGN,' ',FALSE);
```

- **For the AFP Toolbox COBOL Interface:**

This example shows how to use the Set Text Codepage call to convert code points X'D1', X'D2', and X'D3' from codepage T1000850 to codepoints X'63', X'64', and X'65' in output code page T1000875.

```
MOVE "T1000875" TO AFP-CODE-PAGE.
MOVE "HELVETICA CYRILLIC GREEK"
    TO AFP-DESCRIPTIVE-NAME.
MOVE 120 TO AFP-POINT-SIZE.
MOVE MEDIUM TO AFP-WEIGHT.
MOVE NORML TO AFP-FONT-WIDTH.
MOVE ROMAN TO AFP-STYLE.
PERFORM AFPDFNTAT.
MOVE AFP-FONT-ID TO GREEK12.
MOVE TNR12 TO AFP-FONT-ID.
PERFORM AFPSFONT.
MOVE " " TO AFP-SPACE-CHAR.
MOVE "T1000851" TO AFP-CODE-PAGE.
PERFORM AFPSTXCP.
MOVE GREEK12 TO AFP-FONT-ID.
PERFORM AFPSFONT.
PERFORM AFPNLINE.
* Put text on the page
MOVE "JKL"
    TO AFP-CHARACTER-STRING.
MOVE 3 TO AFP-STRING-LENGTH.
MOVE LEFT-ALIGN TO AFP-ALIGNMENT-OPTION.
MOVE FALS TO AFP-POSITION-OPTION.
MOVE -1 TO AFP-EXTRA-VARS-SPACE.
MOVE -1 TO AFP-EXTRA-ICS-SPACE.
MOVE "." TO AFP-ALIGNMENT-CHAR.
PERFORM AFPWRITE.
* EndPage
```

# Set Text Double-Byte Codeset

## Function

Specifies the code page name that is used for encoding input text character strings of DBCS. This function is used to temporarily override the code set requested with Set Input Double-Byte Codeset for the text strings on this page only. See “Set Input Double-Byte Codeset” on page 228 for more information.

Text is automatically translated from the code set specified with Set Text Double-Byte Codeset to the output code set associated with the font when the DBCS text is placed in the output data stream.

## Syntax

### For the AFP Toolbox C Library:

```
AFPSetTextDbCodeset(  
    TBHANDLE    in_pageHandle,  
    char*       in_codeset,  
    char        in_spacechar  
);
```

### For the AFP Toolbox COBOL Library:

```
AFPSTDBCS.  
    CALL "CBLSetTextDbCodeset"  
        USING  
        BY VALUE  
        AFP-PAGE-HANDLE  
        BY CONTENT  
        AFP-CODE-SET  
        AFP-SPACE-CHAR  
        RETURNING  
        AFP-RET-CODE.  
    MOVE "CBLSetTextCP" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

## Input Parameters

### Page handle

The handle for the page. Page handle is returned by the Begin Page call.

### Code set name

The code set name for encoding the character strings in the input character string.

### Space character

Specify a single byte character that is the code point for a blank in the specified code page. This is normally X'20' for ASCII code pages and X'40' for EBCDIC.

## Return Codes

- 4 Page handle is null or is not valid. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

This sample temporarily sets the code set of the current page to IBM-932 for this page only.

- **For the AFP Toolbox C Library:**

```
TBHANDLE docHandle, pageHandle;  
  
AFPBgnPage(docHandle, "This is my current page.", &pageHandle);  
AFPSetTextDbCodeset(pageHandle, "IBM-932", ' ');
```

- **For the AFP Toolbox COBOL Interface:**

## Set Text Double-Byte Codeset

```
MOVE "IBM-932" TO AFP-CODE-SET.  
MOVE SPACES to AFP-SPACE-CHAR.  
PERFORM AFPSTXCP.
```

---

# Set Text Orientation

## Function

This function call moves the text origin and coordinate system for all subsequent text and rule operations on this page. However, it does not affect the positioning or rotation of any image, overlay, bar code, or other data objects or resources.

For example, if the text orientation is set to `I90B180_T0`, the origin is moved 90 degrees clockwise about the page (to the top right hand corner). All subsequent coordinate positions are rotated as well. In effect it is as though the page were rotated 90 degrees counterclockwise and all of the measurements still refer to the top left hand corner. Invalid values are ignored.

**Note:** Use Set Text Orientation to rotate the logical page. Do not use Set Media Size for rotated pages.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPSetTextOrientation(  
    TBHANDLE          in_pageHandle,  
    TextOrientation    in_inlinebaseline  
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPSTO.  
    CALL "CBLSetTextOrientation"  
        USING  
            BY VALUE  
            AFP-PAGE-HANDLE  
            AFP-TEXT-ORIENTATION  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLSetTextOrient" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPSETTXO      PR          10I 0 EXTPROC('AFPSetTextOrientation')  
D AFPPAGHDL      *          * VALUE  
D AFPPAGOR      10I 0 VALUE
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Text orientation

Combination of inline and baseline orientations. See Figure 15 on page 241 for text orientation examples. The valid values are:

#### **I0B90\_T0 (for C), I0B90-T0 (for COBOL), or I0B90T0 (for RPG)**

Text is rotated zero degrees clockwise. The text origin is at the upper, left-hand corner of the page.

#### **I0B270\_T0 (for C), I0B270-T0 (for COBOL), or I0B270T0 (for RPG)**

Text is rotated zero degrees clockwise. The text origin is at the lower, left-hand corner of the page.

#### **I90B180\_T0 (for C), I90B180-T0 (for COBOL), or I90B180T0 (for RPG)**

Text is rotated 90 degrees clockwise. The text origin is at the upper, right-hand corner of the page.



**I90B0\_T0 (for C), I90B0-T0 (for COBOL), or I90B0T0 (for RPG)**

Text is rotated 90 degrees clockwise. The text origin is at the upper, left-hand corner of the page.

**I180B270\_T0 (for C), I180B270-T0 (for COBOL), or I180B270T0 (for RPG)**

Text is rotated 180 degrees clockwise. The text origin is at the lower, right-hand corner of the page.

**I180B90\_T0 (for C), I180B90-T0 (for COBOL), or I180B90T0 (for RPG)**

Text is rotated 180 degrees clockwise. The text origin is at the upper, right-hand corner of the page.

**I270B0\_T0 (for C), I270B0-T0 (for COBOL), or I270B0T0 (for RPG)**

Text is rotated 270 degrees clockwise. The text origin is at the lower, left-hand corner of the page.

**I270B180\_T0 (for C), I270B180-T0 (for COBOL), or I270B180T0 (for RPG)**

Text is rotated 270 degrees clockwise. The text origin is at the lower, right-hand corner of the page.

## Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

Specify that the page should be rotated 90 degrees (landscape):

- **For the AFP Toolbox C Library:**

```
rc = AFPSetTextOrientation(PageHandle, I90B180_T0);
```

- **For the AFP Toolbox COBOL Interface:**

```
rc = CBLSetTextOrientation(PageHandle, I90B180-T0);
```

- **For the AFP Toolbox RPG Interface:**

C	Z-ADD	I90B180T0	AFPPAGOR
C	EXSR	AFPSTXOR	

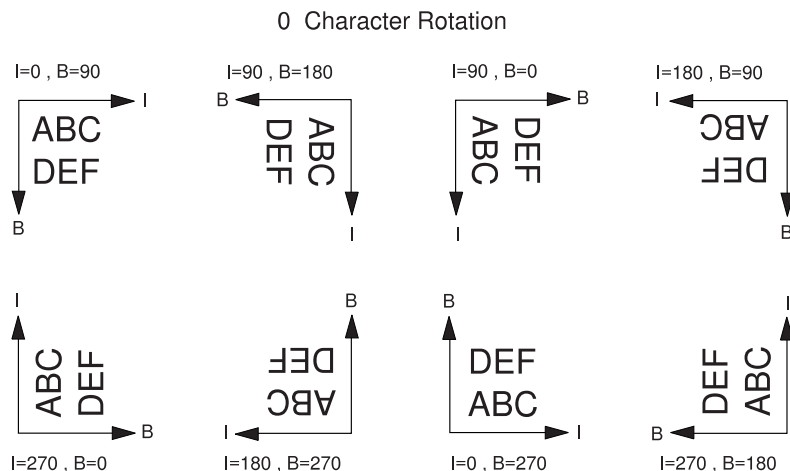


Figure 15. Text Orientation Examples

# Set Units

## Function

This COBOL function sets the units to inches or millimeters for subsequent values, including default values defined in ATXCBVAR. For example, AFP-RULE-THICKNESS has a default value of 12. Without Set Units, that is 12/1440 inch. If the units are changed to MM, that value is 12mm. If the units are changed to INCH, that default is 12 inches, so you should modify the default setting or specify a value for AFP\_RULE\_THICKNESS

You must use integer values when using MM units. When using INCH units, values can have up to 2 decimal places.

The following list shows the performs, functions, and parameters affected by CBLSetUnits:

Perform	Function	Parameters
AFPB CDAT	CBLBarCodeData	X and Y position
AFPB BCD	CBLBgnBarCode	X and Y size
AFPB BOX	CBLBgnBox	width and thickness
AFPB CLR	CBLBgnColor	width
AFPB GOC	CBLBgnGraphic	X and Y size; left, right, bottom, and top edge
AFPB IMG	CBLBgnImage	area X and Y size, X and Y offset
AFPB PARA	CBLBgnPgraph	width and maximum depth
AFPB RULE	CBLBgnRule	thickness
AFPB SHD	CBLBgnShade	width
AFPB TBL	CLGBeginTable	top, bottom, left, and right thickness; table width and depth
AFPC LINK	CBLCreateLink	width and depth
AFPD FLD	CBLDefineField	left and right margin, line spacing
AFPD ROW	CBLDefineRow	top and bottom thickness
AFPG BOX	CBLGDrawBox	horizontal and vertical axes, corner X and Y positions
AFPG CRCL	CBLGCircle	radius
AFPG ELPS	CBLGGllipse	X and Y axes
AFPG PCRC	CBLGPartCircle	radius, X and Y position
AFPG SCP	CBLGSetCharParms	cell width and height
AFPG SLPR	CBLGSetLineParms	line width
AFPH MOVE	CBLHMove	horizontal increment
AFPH MOVETO	CBLHMoveTo	horizontal position
AFPI OBJ	CBLInclObj	width, depth, X and Y offset
AFPP BOX	CBLPutBox	width and depth, rule thickness
AFPP CLR	CBLPutColor	width and depth
AFPH RUL	CBLPutHRule	length and thickness
AFPP SHD	CBLPutShade	width and depth
AFPP VRUL	CBLPutVRule	length and thickness
AFPS MSIZ	CBLSetMediaSize	width and length

AFPSPOS	CBLSetPos	X and Y position
AFPVMOVE	CBLVMove	vertical increment
AFPVMOVETO	CBLVMoveTo	vertical position

In this document, whenever the above parameters are referred to in 1440ths of an inch, you can substitute the units specified by CBLSetUnits.

The following list shows the parameters which are not affected by CBLSetUnits.

AFP-BC-MODWIDTH	PIC S9(4) BINARY VALUE -1.
AFP-BC-ELHEIGHT	PIC S9(4) BINARY VALUE -1.
AFP-EXTRA-VARSPEC	PIC S9(4) BINARY VALUE -1.
AFP-EXTRA-ICSPACE	PIC S9(4) BINARY VALUE -1.
AFP-LINE-SPACING	PIC S9(9) BINARY VALUE 0.

If you override the default values for any of these parameters, you must still use 1440ths of an inch, except for AFP-BC-MODWIDTH which always takes values in 1000ths of an inch.

## Syntax

### For the AFP Toolbox COBOL Interface:

```
AFPSUNI.
  CALL "CBLSetUnits"
    USING
      BY VALUE
        AFP-UNIT-OF-MEASURE
    RETURNING
      AFP-RET-CODE.
  MOVE "CBLSetUnits" TO AFP-ERRDATA.
  PERFORM CHKSUCC.
```

## Input Parameters

### Unit of Measure

The unit of measure you want to use. You can specify INCH (inches), MM (millimeters), or U1440 (to return to the default of 1440 units per inch).

## Return Codes

**99** Units specified were not valid. Specify INCH, MM, or U1440.

## Sample Function Call

This example sets inches to be the unit of measurement for most subsequent values.

### • For the AFP Toolbox COBOL Interface:

```
MOVE INCH TO AFP-UNIT-OF-MEASURE.
PERFORM AFPSUNI.
```

# Translate And Measure Double-Byte String

## Function

Translates the string DBCS characters from **in\_inputCodeSet** to the code set requested when **in\_fontID** font was defined using AFP Define Double-Byte Font By Attribute, then measures the translated string using the metrics from the specified font.

**Note:** You do not need to measure every string that is placed in your document. Measuring a lot of strings has a negative impact on the performance of your program.

## Syntax

```
AFPXlateAndMeasureDbString(  
    FontID      in_fontID,  
    char*       in_inputString,  
    long*       outp_stringWidth,  
    char*       in_inputCodeSet,  
    ushort      in_vinc,  
    ushort      in_icadj,  
    int*        outp_varspaces,  
    int*        outp_icspaces  
);
```

## Input Parameters

### Font ID

The font ID returned by the Define Double-Byte Font By Attribute call.

### Character string

The string of characters, to translate from **in\_inputCodeSet** to the output code page and measure.

### Input code set

Specifies the code set used to encode the input DBCS character string. The defined font must be found with the Define Double-Byte Font By Attribute call.

### Variable increment

The width of the space between words in the specified font measured in 1440ths of an inch. The default value is defined by the font. Specify 0 to use the default value.

### Extra intercharacter space

Amount of extra space added to each intercharacter space (measured in 1440ths of an inch) as the string is measured. Specify 0 to use the default value.

## Output Parameters

### String width

The width of the string in the specified font. String width is returned in units of 1440ths of an inch.

### Variable spaces

A count of the number of spaces between words in the line of text.

### Intercharacter space

A count of the number of character spaces in the line of text.

## Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

## Sample Function Call

Measure the string “hello” in Heisei Gothic font, using input code set IBM-939. Zero is specified for the variable increment and extra intercharacter space parameters, to indicate that the default values should be used. NULL is used as the address of the last two parameters, since we do not care how many characters or spaces were in the string.

```
int rc;
long spwidth;

rc = AFPDefineDbFontByAttr ("T1H01027",      // SBCS codepage
                           "T10300",        // DBCS codepage
                           "IBM-939",        // Codeset
                           "HeiseiKakuGothic", // Font name
                           140,              // Deci-point size
                           0,                // Scale factor
                           MEDIUM_WT,       // Weight
                           MEDIUM_WD,        // Width
                           NORMAL_FS,        // Font style
                           &FontID);         // Font ID

rc = AFPXlateAndMeasureDbString(FontID,      // Font ID
                                "hello",      // DBCS text
                                "IBM-932",    // text's codeset
                                0,            // variable increment
                                0,            // Extrainterchar space
                                &spwidth,     // string width
                                NULL,
                                NULL);
```

# Translate And Measure String

## Function

Translates the string of characters from **in\_inputCodePage** to the code page requested when **in\_fontID** font was defined, then measures the translated string using the metrics from the specified font.

**Note:** You do not need to measure every string that is placed in your document. Measuring a lot of strings has a negative impact on the performance of your program.

## Syntax

### For the AFP Toolbox C Library:

```
AFPXlateAndMeasureString(  
    FontID      in_fontID,  
    char*       in_inputString,  
    long*       outp_stringWidth,  
    char*       in_inputCodePage,  
    ushort      in_varinc,  
    ushort      in_icharspace,  
    int*        outp_varspaces,  
    int*        outp_icspaces  
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPXMSTR.  
    CALL "CBLXlateAndMeasureString"  
        USING  
            BY VALUE  
                AFP-FONT-ID  
            BY CONTENT  
                AFP-CHARACTER-STRING  
            BY REFERENCE  
                AFP-MEASURED-WIDTH  
            BY CONTENT  
                AFP-CODE-PAGE  
            BY VALUE  
                AFP-EXTRA-VARSPACE  
                AFP-EXTRA-ICSPACE  
            BY REFERENCE  
                AFP-VARSPACES  
                AFP-ICSPACES  
        RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLXlateMeasureStr" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPXLMSTR      PR          101 0  EXTPROC('AFPXlateAndMeasureString')  
D  AFPFONTID          51 0  VALUE  
D  AFPCHRSTR          250  
D  AFPSTRWID          101 0  
D  AFPCDEPAG           9  
D  AFPXTRVAR          51 0  VALUE  
D  AFPXTRICS          51 0  VALUE  
D  AFPVARCNT          101 0  VALUE  
D  AFPICSCNT          101 0  VALUE
```

## Input Parameters

### Font ID

The font ID returned by the Define Font By Attribute or Define Font By Name call.

### Character string

The string of characters to translate from **in\_inputCodePage** to the output code page and measure.

### Input code page

Up to eight characters, specifies the code page used to encode the input character string. If more than eight are specified, only the first eight characters are used.

### Variable increment

The width of the space between words in the specified font measured in 1440ths of an inch. The default value is defined by the font. Specify 0 to use the default value.

### Extra intercharacter space

Amount of extra space to add to each intercharacter space (measured in 1440ths of an inch) as the string is measured. Specify 0 to use the default value.

## Output Parameters

### String width

The width of the string in the specified font. String width is returned in units of 1440ths of an inch.

### Variable spaces

A count of the number of spaces between words in the line of text.

### Intercharacter spaces

A count of the number of character spaces in the line of text.

## Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Measure the string “hello” in Helvetica font, using input code page 850 and output code page 500. Zero is specified for the variable increment and extra intercharacter space parameters, to indicate that the default values should be used. NULL is used as the address of the last two parameters, since we do not care how many characters or spaces were in the string.

```
FontID textfont;
long spwidth;
char inputcp[]="T1000850";
char outputcp[]="T1V10500";
char tempchar[]="hello";
AFPDefineFontByAttr(outputcp,"Helvetica Latin1",100,MEDIUM_WT,
                    MEDIUM_WD,NORMAL_FS,&textfont);
AFPXlateAndMeasureString(textfont,tempchar,inputcp,
                        0,0,&spwidth,NULL,NULL);
```

- **For the AFP Toolbox COBOL Interface:**

This example measures the characters in the font character set assigned to code points X'61', X'62', and X'63' (lowercase a, b, and c characters in ASCII) from code page 850.

```
* AFPXlateAndMeasureString
* x'61 62 63 is ascii lowercase a b c
* in codepage 500 these characters are x'81 82 83
* in the font these characters measure 56 240th pels
  03 HEXSTRING PIC XXX VALUE x'616263'
  MOVE TNRI2 TO AFP-FONT-ID.
  MOVE "HEXSTRING" TO AFP-CHARACTER-STRING.
  MOVE "T1000850" TO AFP-CODE-PAGE.
```

## Translate and Measure String

```
PERFORM AFPXMSTR.  
* After the function call,  
* AFP-MEASURED-WIDTH should hold 336  
* AFP-VARSPPACES      should hold 0 (# of blanks)  
* AFP-ICSPACES        should hold 2 (# of candidates  
*   for intercharacter space  
    DISPLAY AFP-MEASURED-WIDTH UPON SYSOUT.  
    DISPLAY AFP-VARSPPACES    UPON SYSOUT.  
    DISPLAY AFP-ICSPACES      UPON SYSOUT.
```

- **For the AFP Toolbox RPG Interface:**

This example measures the characters in the font character set assigned to code points X'61', X'62', and X'63' (lowercase a, b, and c characters in ASCII) from code page 850.

```
C          Z-ADD      TNR12      AFPFNTID  
C          EXSR      X'616263'   AFPCHRSTR
```



## Vertical Move

### Function

Move vertically, that is, move the baseline position, relative to the current position. Do not issue this call within a paragraph or a table.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPVMove(
    TBHANDLE    in_pageHandle,
    short       in_incrementValue
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPVMOVE.
    CALL "CBLVMove"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-Y-COORDINATE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLVMove" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPVMOVE      PR          10I 0 EXTPROC('AFPVMove')
D AFPPAGHDL      *      VALUE
D AFPYCRD      10I 0 VALUE
```

## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Increment value

The amount of the vertical move. If the value is negative, the move is up, otherwise it is down.

## Return Codes

- 4** Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 14** This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.

## Sample Function Call

Add two inches to the current vertical position:

- **For the AFP Toolbox C Library:**

```
VDist = IN_2_U1440(2);
rc = AFPVMove(PageHandle, VDist);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE 2880 TO AFP-Y-COORDINATE.
PERFORM AFPVMOVE.
```

- **For the AFP Toolbox RPG Interface:**

## Vertical Move

C	Z-ADD	2880	AFPYCRD
C	EXSR	AFPVMOVER	

## Vertical Move To

### Function

Move vertically, that is, move the baseline position, to a specified position. The new current position is at the specified position (baseline) and unchanged in the inline direction. Do not issue this call within a paragraph or a table.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPVMoveTo(
    TBHANDLE    in_pageHandle,
    short       in_baselinePosition,
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPVMOVETO.
    CALL "CBLVMoveTo"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
                AFP-Y-COORDINATE
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLVMoveTo" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

**For the AFP Toolbox RPG Interface:**

```
DAFPVMOVTO      PR          10I 0 EXTPROC('AFPVMoveTo')
D AFPPAGHDL      *          *  VALUE
D AFPYCRD        10I 0 VALUE
```

### Input Parameters

#### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

#### Baseline Position

The new vertical position.

### Return Codes

- 4** Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 14** This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.

### Sample Function Call

Set the current vertical position to two inches:

- **For the AFP Toolbox C Library:**

```
VDist = IN_2_U1440(2);
rc = AFPVMoveTo(PageHandle, VDist);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE 2880 TO AFP-Y-COORDINATE.
PERFORM AFPVMOVETO.
```

## Vertical Move To

- **For the AFP Toolbox RPG Interface:**

C	Z-ADD	2880	AFPYCRD
C	EXSR	AFPVMOVTOR	

## Write Double-Byte String

### Function

Places a combination of SBCS and DBCS text in the current page at the current position.

**Write Double-Byte String** places the entire character string at the current position and does not try to break the string into multiple output lines. If the string is too wide or too deep to fit on the page at the current position, an error return code is returned by Write Double-Byte String, but the string is placed anyway. This might result in errors at print time.

**Note:** If the string is printed with zero width blanks, AFP Toolbox was probably unable to find the correct font metric information. For more information about locating the correct font metric information, see the trouble shooting section in the appendix for your environment.

### Syntax

**For the AFP Toolbox C Library:**

```
int AFPWriteDbString(
    TBHANDLE          in_currentHandle,
    char*              in_inputString,
    short              in_variableIncrement,
    short              in_intercharacterSpacing,
    AlignmentOption    in_alignment_option,
    char*              in_alignment_charptr,
    boolean            in_move_pos
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPWRDBCS.
    CALL "CBLWriteDbString"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
            BY CONTENT
                AFP-CHARACTER-STRING
            BY VALUE
                AFP-EXTRA-VARSPACE
                AFP-EXTRA-ICSPACE
                AFP-ALIGNMENT-OPTION
                AFP-ALIGNMENT-CHAR
                AFP-POSITION-OPTION
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLWriteDbString" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### Current handle

The handle for the current document or page returned from the Begin Document or the Begin Page call. If the page handle is for the document, a group level tag is built. Otherwise, a page level tag is built.

#### Character string

The string of characters to write (null terminated).

#### Variable increment

The width of a blank used for this string. The keyword DEFAULT\_WS can be used to indicate that the width of a blank should be determined by the current font.

## Write Double-Byte String

### Intercharacter increment

The amount of extra space inserted between characters in this string. The keyword DEFAULT\_ICS can be used to indicate that the width of a blank should be determined by the font.

### Alignment option

Specifies how the string of text should be aligned at the current location. Valid values are:

```
LEFT_ALIGN      /* left aligned */
RIGHT_ALIGN     /* right aligned */
CENTER_ALIGN    /* centered */
CHAR_ALIGN      /* align on 1st occurrence of character */
LAST_CHAR_ALIGN /* align on last occurrence of character */
```

### Alignment character

If a DBCS character is specified for the alignment option, this parameter gives the character on which to align the string. For example, you might line up a column of numbers on the decimal point.

### Move position

Indicates whether the current position is updated at the conclusion of this function. If this parameter is set, the current position remains at the origin of the string. Otherwise, the current position is moved to the position at which the next character would be placed.

**Note:** For best performance results, specify TRUE.

## Return Codes

- 4      Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 15     Set Font must be issued before calling Write Text Double-Byte String.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Print a string of DBCS characters aligned with a character "M" in the string:

```
TBHANDLE toolboxSession;
TBHANDLE outputDocument;

AFPBgnSession(&toolboxSession);
AFPBgnDoc(toolboxSession, outputDocName, outputDocComment,
          TO_OUTPUT_FILE, outputDocFileName, NULL, &outputDocument);

AFPSetInputDbCodeset(toolboxSession, "IBM-932", ' ');
AFPSetFontType(FONT_OLN);

AFPDefineDbFontByAttr("T1H01027", "T10300", "IBM-939",
                     "HeiseiMincho", 100, 0,
                     LIGHT_WT, MEDIUM_WD, NORMAL_FS,
                     &hFont);

AFPBgnPage(outputDocument, "My Next Page", &page);
AFPWriteDbString(page, "Enter DBCS text string",
                 -1, -1,
                 CHAR_ALIGN, "M",
                 TRUE);
```

- **For the AFP Toolbox COBOL Interface:**

Print a string of characters centered at the current location, using default character and word spacing:

**Note:** You must set the length of the string in AFP-STRING-LENGTH before calling this function. You can use the TRIM and STRINGLEN functions provided in MVS and OS/400 to calculate this length. See the appendix for your operating system for more information.

```

*BeginPage
  PERFORM AFPBPAGE.
  MOVE "T1H01027" TO AFP-CODE-PAGE.
  MOVE "T10300" TO AFP-CHAR-SET.
  PERFORM AFPDFNTNM.
  MOVE AFP-FONT-ID TO MYFONT.
  MOVE MYFONT TO AFP-FONT-ID.
  PERFORM AFPSFONT.
* SetPos to lin,lin
  MOVE 1440 TO AFP-X-COORDINATE.
  MOVE 1440 TO AFP-Y-COORDINATE.
  MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
  MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
  PERFORM AFPSPOS.
  MOVE "This is text" TO AFP-CHARACTER-STRING.
  MOVE 12 TO AFP-STRING-LENGTH.
  MOVE CENTER-ALIGN TO AFP-ALIGNMENT-OPTION.
  MOVE FALS TO AFP-POSITION-OPTION.
  PERFORM AFPWRDBCS .
* EndPage
  PERFORM AFPEPAGE.

```

---

# Write String

## Function

Places text in the current page at the current position.

**Write String** places the entire character string at the current position and does not try to break the string into multiple output lines. If the string is too wide or too deep to fit on the page at the current position, an error return code is returned by Write String, but the string is placed anyway. This might result in errors at print time.

**Note:** If the string is printed with zero width blanks, AFP Toolbox was probably unable to find the correct font metric information. For more information about locating the correct font metric information, see the troubleshooting section in the appendix for your environment.

## Syntax

### For the AFP Toolbox C Library:

```
int AFPWriteString(
    TBHANDLE          in_pageHandle,
    char*              in_inputString,
    short              in_variableIncrement,
    short              in_intercharacterSpacing,
    AlignmentOption    in_alignment_option,
    char               in_alignment_char,
    boolean            in_same_pos
);
```

### For the AFP Toolbox COBOL Interface:

```
AFPWRITE.
    CALL "CBLWriteString"
        USING
            BY VALUE
                AFP-PAGE-HANDLE
            BY CONTENT
                AFP-CHARACTER-STRING
            BY VALUE
                AFP-EXTRA-VARSPACE
                AFP-EXTRA-ICSPACE
                AFP-ALIGNMENT-OPTION
                AFP-ALIGNMENT-CHAR
                AFP-POSITION-OPTION
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLWriteString" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### For the AFP Toolbox RPG Interface:

```
DAFPWRITE      PR          10I 0 EXTPROC('AFPWriteString')
D AFPPAGHDL          *    VALUE
D AFPCHRSTR          250
D AFPXTRVAR          5I 0 VALUE
D AFPXTRICS          5I 0 VALUE
D AFPALNOPT          10I 0 VALUE
D AFPALNCHR          1    VALUE
D AFPPOSOPT          10I 0 VALUE
```



## Input Parameters

### Page handle

The handle for the current page. Page handle is returned by the Begin Page call.

### Character string

The string of characters to write (null terminated).

### Variable increment

The width of a blank used for this string. The keyword DEFAULT\_WS can be used to indicate that the width of a blank should be determined by the current font.

### Intercharacter increment

The amount of extra space inserted between characters in this string. The keyword DEFAULT\_ICS can be used to indicate that the width of a blank should be determined by the font.

### Alignment option

Specifies how the string of text should be aligned at the current location. The valid values are:

```
LEFT_ALIGN      /* left aligned */
RIGHT_ALIGN     /* right aligned */
CENTER_ALIGN    /* centered */
CHAR_ALIGN      /* align on 1st occurrence of character */
LAST_CHAR_ALIGN /* align on last occurrence of character */
```

### Alignment character

If a character is specified for the alignment option, this parameter gives the character on which to align the string. For example, you might line up a column of numbers on the decimal point.

### Same position

Indicates whether the current position is updated at the conclusion of this function. If this parameter is set to TRUE, the current position remains at the origin of the string. Otherwise, the current position is moved to the position at which the next character would be placed.

**Note:** For best performance results, specify TRUE.

## Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 15 Set Font must be issued before calling Write String.

## Sample Function Call

- **For the AFP Toolbox C Library:**

Print a string of characters centered at the current location, using default character and word spacing:

```
strcpy(CharString,"Here is my text to center");
rc = AFPWriteString(PageHandle, CharString, DEFAULT_WS,
                   DEFAULT_ICS, CENTER_ALIGN, 0, TRUE);
```

- **For the AFP Toolbox COBOL Interface:**

Print a string of characters centered at the current location, using default character and word spacing:

**Note:** You must set the length of the string in AFP-STRING-LENGTH before calling this function. You can use the TRIM and STRINGLEN functions provided in MVS and OS/400 to calculate this length. See the appendix for your operating system for more information.

```
*BeginPage
PERFORM AFPBPAGE.
MOVE "T1V10500" TO AFP-CODE-PAGE.
MOVE "C0N20000" TO AFP-CHAR-SET.
PERFORM AFPDFNTNM.
MOVE AFP-FONT-ID TO MYFONT.
```

## Write String

```
        MOVE MYFONT    TO AFP-FONT-ID.
        PERFORM AFPSFONT.
* SetPos to lin,lin
        MOVE 1440    TO AFP-X-COORDINATE.
        MOVE 1440    TO AFP-Y-COORDINATE.
        MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
        MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
        PERFORM AFPSPOS.
        MOVE "This is text" TO AFP-CHARACTER-STRING.
        MOVE 12 TO AFP-STRING-LENGTH.
        MOVE CENTER-ALIGN TO AFP-ALIGNMENT-OPTION.
        MOVE FALS TO AFP-POSITION-OPTION.
        PERFORM AFPWRITE.
* EndPage
        PERFORM AFPEPAGE.
```

- **For the AFP Toolbox RPG Interface:**

Print a string of characters left-aligned at the current location, using default character and word spacing:

```

C      DTA(2)      CAT(P)  X'00':0    AFPCHRSTR
C      MOVE      LEFTALGN  AFPALNOPT
C      MOVE      FALS      AFPPOSOPT
C      Z-ADD      -1        AFPXTRVAR
C      Z-ADD      -1        AFPXTRICS
C      MOVE      ' .'      AFPALNCHR
C      EXSR      AFPWRITER
...
**CTDATA DTA
HELLO WORLD
```

2

---

## Font Return Codes

This section contains a listing of all of the font return codes for the AFP Toolbox.

- 1** The specified code page file could not be found. The font is still usable, however, no code page translations can be performed. Unless the data stream code page and default input code page are identical, incorrect results will occur.
- 2** The AFM file specified in the font index could not be found. Run FLIPLIST to find the name of the character set and determine if the file exists and has appropriate read permissions.
- 3** The character set file specified in the font index or Define Font By Name call could not be found. Check the font listing for your environment to find the name of the character set and determine if the file exists and has appropriate read permissions.
- 4** Neither the character set file nor the corresponding AFM file could be found. The font is still usable; however, no metrics are available, thus right, center, and character aligned text, next line operations, and any other operation involving measured text will produce incorrect output.
- 5** This code page is SBCS, specify a DBCS code page.
- 6** This code page is DBCS, specify an SBCS code page.
- 7** Required parameter was NULL. One or more of the required parameters was incorrectly specified as NULL.
- 9** Invalid file. The code page file referenced by the specified code page name is not valid. If you are running on an MVS system, ensure PTF UQ08195 is applied.
- 1** The AFP font index file could not be found. See the appendix for your operating system for information about where to find the font index.
- 2** The font family name was not found in the font table. Run FLIPLIST to see if the family name is correctly specified.
- 3** The combination of point size, weight, width, and attributes was not defined with the specified font family. Check the font listing to see if the font particulars were correctly specified.
- 4** The font character set was not defined within the specified font family or the character set was not found in the font index. Check the font listing to see if the font character set was correctly specified.
- 5** The font FGID/GCSGID combination was not defined within the specified font family. Check the font listing to see if the font FGID/GCSGID combination was correctly specified.
- 6** The font **i** and **j** indexes were found, but the internal font information could not be found in the font tables. This is an internal error that could be caused by memory corruption. Report this error to IBM Service.
- 7** There are no font handles available to define a new font. The maximum number of fonts that can be defined is 1024. If you get this return code when your program is creating multiple sessions or documents by looping, you need to delete font definitions each time through the loop. See "Delete Font" on page 131 for details.
- 8** The code page file specification is missing on the define font operation. Check to ensure that a valid code page file was passed as a parameter on one of the Define Font calls.
- 9** The font handle specified is not valid. Font handles must be obtained from a successful call to one of the Define Font calls.
- 10** A NULL or 0 value was passed as a required parameter. Pass a non-NULL value of the correct type for this parameter.
- 12** The specified code page is not valid. The file in question is probably an AFP character set, not a code page.

## Font Return Codes

- 13 The specified code page is not valid. The file in question is probably an AFP document file, not a code page.
- 14 The specified code page is not valid. The contents of the code page are incorrect. Use the AFPDMP utility to display the file contents and determine if the file is in error. If you are running on an MVS system, ensure that PTF UQ08195 has been applied.
- 15 The input code page file on a translate operation could not be found.
- 16 The output code page file on a translate operation could not be found.
- 17 The specified font type is not valid. The value specified for font type must be one of the enumeration values defined in the FontCSType data type.
- 18 The specified code page is not valid. The first code page must be SBCS and the second code page DBCS.

---

## Chapter 4. AFP Toolbox GOCA Functions

This section describes the functions provided with AFP Toolbox for building Graphic Objects (GOCA). This function requires PTFs to be applied for your environment. Not all GOCA functions are supported by all AFP printers. GOCA functions are provided for C++, C, and COBOL languages.

A brief overview of GOCA is provided here. For a more thorough description of GOCA objects and structures, see the *Graphics Object Content Architecture Reference*.

**Note:** GOCA functions are not supported on the OS/400.

---

### Graphic Presentation Space Concepts

Within a GOCA object, positioning is performed relative to the coordinates in the Graphic Presentation Space (GPS), as shown in Figure 16. In order to create a graphic object, you must first set the AFP Toolbox's position using any of the existing positioning commands in your program (shown in the figure as GOCA origin). Once you have started the GOCA object, any positioning on the Graphic commands are done using GPS coordinates. The AFP Toolbox creates a GPS space for you that has the (0,0) origin centered within the rectangle defined by the width and depth parameters that you specify on the Begin Graphic command.

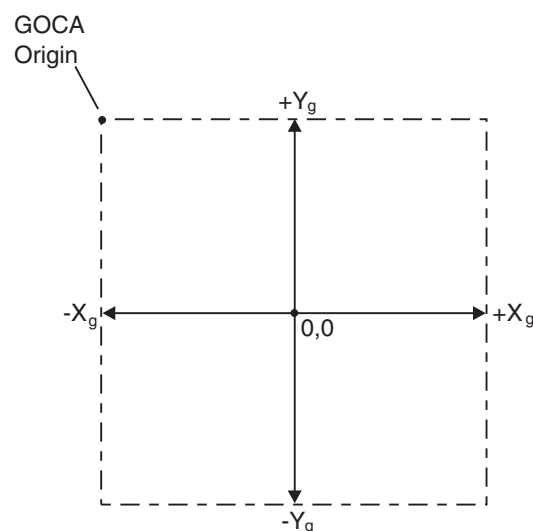


Figure 16. Graphics Presentation Space

---

### C++ Objects

See “Graphic Object Content Architecture (GOCA)” on page 346 for information about GOCA C++ objects.

---

### C and COBOL Function Descriptions

Each function call description includes the following sections:

#### Function

A description of the major purpose of each function.

#### Syntax

A diagram showing the function parameters.

**Input Parameters**

Explanation of each input parameter.

**Output Parameters**

Explanation of each output parameter.

**Return Codes**

List of possible return codes (other than zero, which is normal).

**Sample Function Call**

Provides sample code for using the function. All sample functions assume that prerequisite calls and variable definitions have been made prior to the call. In other words, only the actual function is shown and not everything that would need to be coded before the function call.

When calling these functions, *every parameter must be specified and given in the order shown in the diagram.*

---

## Header files and Copy Books

C functions are defined in the main toolbox header file **aocl.h** which must be included in your program.

COBOL perform statements are in ATXCBGPR and ATXCBPRF copy books. COBOL variables are in ATXCBGVA and ATXCBVAR copy books.

See the appendix for your operating system for information about how to find these files at your installation.

## Begin Graphic Object

### Function

Begins a graphic object in the specified page. Other graphic drawing order functions as described in this chapter can be included in the graphic object. The object is placed in the page when the End Graphic Object function is called.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPBgnGraphic (TBHANDLE      in_pageHandle,
               char*          in_GOCAName,
               ulong          in_areaxsize,
               ulong          in_areaysize,
               short          in_areaxlw,
               short          in_areaylw,
               short          in_areaybw,
               short          in_areaytw,
               MappingOption in_objmapopt,
               DegRot         in_objrotation,
               TBHANDLE*      outp_gocaHandle
               );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPBGOC.
  CALL      "CBLBgnGraphic"
           USING
             BY VALUE
               AFP-PAGE-HANDLE
             BY CONTENT
               AFP-GOCA-NAME
             BY VALUE
               AFP-AREA-XSIZE
               AFP-AREA-YSIZE
               AFP-AREA-XLW
               AFP-AREA-XRW
               AFP-AREA-YBW
               AFP-AREA-YTW
               AFP-OBJECT-MAPPING-OPTION
               AFP-OBJECT-ROTATION
             BY REFERENCE
               AFP-GOCA-HANDLE
           RETURNING
             AFP-RET-CODE.
  MOVE "CBLBgnGraphic" TO AFP-ERRDATA.
  PERFORM CHKSUCC.
```

## Input Parameters

### Page Handle

The handle for the page to contain the graphic object. Page handle is returned by the Begin Page call.

### GOCA Name

Up to 252 characters to be used as the name of this object in the output data stream. The first eight characters are used in the Begin Graphic structured field as the graphic object name. If more than eight characters are specified, an FQN triplet will also be built to contain the entire string.

### Area X Size, Area Y Size

These parameters specify the size of the area (presentation space) for the graphic object. **areaxsize** and **areaysize** are the width and depth of the space, respectively.

## Begin Graphic Object

### Area XLW, XRW, YBW, and YTW

These parameters give the left, right, bottom, and top edges, respectively, of the graphics presentation space (GPS). Keep in mind that the GPS is a Cartesian coordinate system with the origin at (0,0) in the center, and not at the upper left hand corner like an AFP page, as shown in Figure 17.

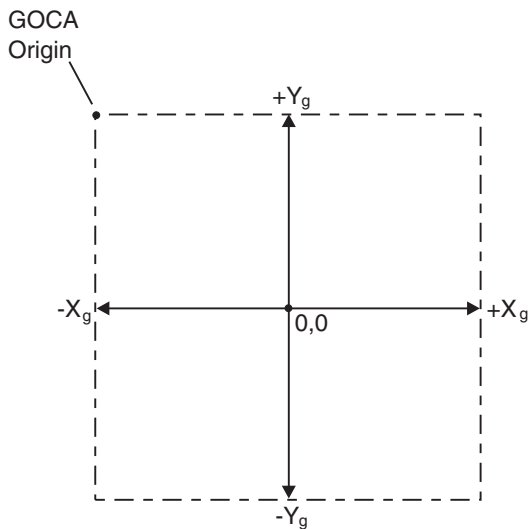


Figure 17. Graphics Presentation Space

### Mapping Option

Get **DegRot** definition from Include Object.

### Rotation

Specifies the clockwise rotation of the object area from the Xaxis rotation of the page. Valid values are:

#### **DEG\_0 (for C) or DEG-0 (for COBOL)**

No rotation is applied to the object area.

#### **DEG\_90 (for C) or DEG-90 (for COBOL)**

The object area is rotated 90 degrees in the clockwise direction.

#### **DEG\_180 (for C) or DEG-180 (for COBOL)**

The object area is rotated 180 degrees in the clockwise direction.

#### **DEG\_270 (for C) or DEG-270 (for COBOL)**

The object area is rotated 270 degrees in the clockwise direction.

## Output Parameters

### GOCA Handle

Handle to be used on subsequent graphic functions to place the actual drawing orders into the object. The handle is also used by the End Graphic object call.

## Return Codes

- 1 The **Graphic Handle** value is null. You must specify a valid address for this parameter.
- 4 Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.
- 14 This function is not allowed inside a paragraph. Issue End Paragraph before issuing this call.



## Sample Function Call

- **For the AFP Toolbox C Library:**

This example creates a graphic object 100 mm square with the upper left-hand corner of the presentation space at (80mm, 40mm) as determined by the Set Position command. A circle is drawn inside the graphic object with a diagonal fill pattern and a radius of 10mm.

```
rc = AFPSetPos (PageHandle,MM_2_U1440(80),ABSOLUTE_POS,
               MM_2_U1440(40),ABSOLUTE_POS) ;
rc = AFPBgnGraphic(PageHandle,"C GOCA",
                   MM_2_U1440(100), MM_2_U1440(100),
                   MM_2_U1440(-50),MM_2_U1440(50),
                   MM_2_U1440(-50),MM_2_U1440(50),POSITION_MOPT,
                   DEG_0,&GrHandle);
rc = AFPGSetPatternSym(GrHandle,BLTR2_FP) ;
rc = AFPGSetGPSPos(GrHandle,0,0);
rc = AFPGBgnArea(GrHandle,TRUE);
rc = AFPGCircle(GrHandle,MM_2_U1440(10));
rc = AFPGEndArea(GrHandle,1);
rc = AFPEndGraphic(PageHandle,&GrHandle);
```

- **For the AFP Toolbox COBOL Interface:**

This example creates a graphic object four in square with the upper left-hand corner of the presentation space at (1 in, 2 in) as determined by the AFPSPPOS command. A circle is drawn inside the graphic object with a diagonal fill pattern and a radius of one inch.

```
MOVE 1440 TO AFP-X-COORDINATE.
MOVE 2880 TO AFP-Y-COORDINATE.
MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
PERFORM AFPSPPOS.
MOVE 5760 TO AFP-AREA-XSIZE.
MOVE 5760 TO AFP-AREA-YSIZE.
MOVE -2880 TO AFP-AREA-XLW.
MOVE 2880 TO AFP-AREA-XRW.
MOVE -2880 TO AFP-AREA-YBW.
MOVE 2880 TO AFP-AREA-YTW.
MOVE DEG-0 TO AFP-OBJECT-ROTATION.
MOVE POSITION-AND-TRIM TO AFP-OBJECT-MAPPING-OPTION.
PERFORM AFPBGOC.
MOVE BLTR2-FP TO AFP-PATTERN.
PERFORM AFPGSPSM.
MOVE TRU TO AFP-DRAW-BOUNDARY.
PERFORM AFPGBGA.
MOVE 1440 TO AFP-RADIUS.
PERFORM AFPGCRCL.
PERFORM AFPGEGA.
PERFORM AFPEGOC.
```

# End Graphic Object

## Function

Begins a graphic object in the specified page. Other graphic drawing order functions as described in this chapter can be included in the graphic object. The object is placed in the page when the End Graphic Object function is called.

## Syntax

**For the AFP Toolbox C Library:**

```
AFPEndGraphic (TBHANDLE      in_pageHandle,  
               TBHANDLE*     inp_gocaHandle  
               );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPEGOC.  
  CALL      "CBLEndGraphic"  
          USING  
          BY VALUE  
          AFP-PAGE-HANDLE  
          BY REFERENCE  
          AFP-GOCA-HANDLE  
          RETURNING  
          AFP-RET-CODE.  
  MOVE "CBLEndGraphic" TO AFP-ERRDATA.  
  PERFORM CHKSUCC.
```

## Input Parameters

### Page Handle

The handle for the page to contain the graphic object. Page handle is returned by the Begin Page call.

### GOCA Handle

Handle returned by the Begin Graphic call. This parameter is set to null by the End Graphic call.

## Return Codes

- 1      The **Graphic Handle** value is null. You must specify a valid address for this parameter.
- 4      Page handle is not pointing to a valid page. Make sure you have called Begin Page before issuing this call.

## Sample Function Call

See “Begin Graphic Object” on page 263 for an entire graphics example.

## GOCA Begin Area

### Function

Begins an area inside a GOCA object. Areas are two-dimensional primitives which are filled with a pattern. The only GOCA orders that are allowed inside an area are:

- Comment
- Draw Box
- Fillet
- Full Arc
- Line
- No-op
- Relative Line
- Set Arc Parameters
- Set Extended Color
- Set GPS Position
- Set Line Parm

The Toolbox does not return an error if you try to use other orders inside your graphic object, but the printer will.

This drawing order does not change the current GPS position.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPGBgnArea (TBHANDLE      in_gocaHandle,
              boolean       in_drawbound
              );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGBGA.
    CALL "CBLGBgnArea"
        USING
            BY VALUE
                AFP-GOCA-HANDLE
                AFP-DRAW-BOUNDARY
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLGBgnArea" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

#### Draw boundary

Boolean value to indicate whether the boundary of the area should be drawn by the printer.

### Return Codes

none

## GOCA Begin Area

### Sample Function Call

Start an area with the boundary drawn:

- **For the AFP Toolbox C Library:**

```
rc = AFPGBgnArea(GrHandle,TRUE);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE TRU TO AFP-DRAW-BOUNDARY.  
PERFORM AFPGBGA.
```

## GOCA Character String

### Function

Place a string of text in a GOCA object. The string is formatted in the last font specified with **AFPSetFont (GOCA Set Font)** or the device default font for GOCA if no **AFPSetFont** has been performed. The string is always left-aligned at the current GPS position. The string is translated to the codepage specified on the font definition.

This drawing order does not change the current GPS position.

**Note:** Some older printers do not handle GOCA text correctly. If the text does not format as you think it should, place the character string into your page using **AFPWrite** instead of placing it inside the GOCA object.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPCharStr (TBHANDLE      in_gocaHandle,
            char*          in_drawbound
            );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPCharStr.
  ADD 1 TO AFP-STRING-LENGTH.
  MOVE LOW-VALUES
    TO AFP-CHARACTER-STRING(AFP-STRING-LENGTH:1).
  CALL "CBLGCharStr"
    USING
      BY VALUE
        AFP-GOCA-HANDLE
      BY CONTENT
        AFP-CHARACTER-STRING
    RETURNING
      AFP-RET-CODE.
  MOVE "CBLGCharStr" TO AFP-ERRDATA.
  MOVE SPACES TO AFP-CHARACTER-STRING(AFP-STRING-LENGTH:1).
  PERFORM CHKSUCC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the **Begin Graphic** call.

#### Character string

The string of characters that you want to be printed.

**Note:** For COBOL you must also set the value of **AFP-STRING-LENGTH** to be the number of characters in the string.

### Return Codes

**none**

## GOCA Character String

### Sample Function Call

Print a character string at the current GPS position:

- **For the AFP Toolbox C Library:**

```
rc = AFPGCharStr(GrHandle,"Text inside the GOCA");
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE 20 TO AFP-STRING-LENGTH.  
MOVE "Text inside the GOCA" TO AFP-CHARACTER-STRING.  
PERFORM AFPGCST.
```

## GOCA Circle

### Function

Draw a circle at the current GPS position. The circle is filled with a pattern if it is inside a graphics area and you have specified a fill pattern with GOCA Set Pattern Symbol.

This drawing order does not change the current GPS position.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPGCircle (TBHANDLE      in_gocaHandle,
            short          in_radius
            );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGCRCL.
  CALL    "CBLGCircle"
        USING
        BY VALUE
        AFP-GOCA-HANDLE
        AFP-RADIUS
        RETURNING
        AFP-RET-CODE.
  MOVE "CBLGCircle" TO AFP-ERRDATA.
  PERFORM CHKSUCC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

#### Radius

The length, in units of 1440ths of an inch, of the radius (distance from the center to the edge) of the circle you want drawn.

### Return Codes

none

### Sample Function Call

See “Begin Graphic Object” on page 263 for an example of drawing a circle.

# GOCA Draw Box

## Function

Draw a box using the current GPS position as one corner and the specified X and Y coordinates as the other. The box can have square or rounded corners. The box is filled with a pattern if it is inside a graphics area and you have specified a fill pattern with GOCA Set Pattern Symbol.

This drawing order does not change the current GPS position.

**Note:** This drawing order is not supported on all IBM printers.

## Syntax

### For the AFP Toolbox C Library:

```
AFPGDrawBox (TBHANDLE      in_gocaHandle,  
             short          in_cornerXpos,  
             short          in_cornerYpos,  
             ushort         in_haxisval,  
             ushort         in_vaxisval  
             );
```

### For the AFP Toolbox COBOL Interface:

```
AFPGBOX.  
  CALL      "CBLGDrawBox"  
          USING  
          BY VALUE  
          AFP-GOCA-HANDLE  
          AFP-CORNER-XPOS  
          AFP-CORNER-YPOS  
          AFP-HAXISVAL  
          AFP-VAXISVAL  
          RETURNING  
          AFP-RET-CODE.  
  MOVE "CBLGDrawBox" TO AFP-ERRDATA.  
  PERFORM CHKSUCC.
```

## Input Parameters

### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

### Corner X and Y Positions

The position in the GPS for the opposite corner of the box. The box is drawn from the current position to this (X,Y) position.

### Horizontal and Vertical Axis Values

These values specify the axes of a full ellipse. A quarter of the ellipse is used to draw each corner of the box. If the horizontal and vertical values are equal, quadrants of a circle are used instead of an ellipse. If both values are zero then a square-cornered box is drawn.

## Return Codes

none



## Sample Function Call

Draw a box with rounded corners from the current position to (1 in, 1 in).

- **For the AFP Toolbox C Library:**

```
rc = AFPGDrawBox(GrHandle, 1440, 1440, 100, 140);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE 1440 TO AFP-CORNER-XPOS.  
MOVE 1440 TO AFP-CORNER-YPOS.  
MOVE 100 TO AFP-HAXISVAL.  
MOVE 140 TO AFP-VAXISVAL.  
PERFORM AFPGBOX.
```

## GOCA Draw Objects

---

### Function

This function is used to draw a series of drawing orders from the current location. The objects can be fillets, markers, lines, or relative lines. The positions for the objects are supplied as an array of integers, consisting of the X position followed by the Y position. For example, array position zero would contain the X coordinate and array position one would contain the Y coordinate for the first object, array positions two and three would contain the X and Y coordinate for the second object, and so forth.

The current GPS position is set to the last of the (X,Y) coordinate pairs that are specified.

### Syntax

#### For the AFP Toolbox C Library:

```
AFPGDrawObjects (TBHANDLE      in_gocaHandle,
                  GOCATYPE      in_objtype,
                  short          in_count,
                  short*         in_xypairarray
                  );
```

#### For the AFP Toolbox COBOL Interface:

```
AFPGOBSJ.
  CALL      "CBLGDrawObjects"
           USING
           BY VALUE
             AFP-GOCA-HANDLE
             AFP-GOCAOBJ-TYPE
             AFP-XYCOUNT
           BY CONTENT
             AFP-XYPAIRS-ARRAY
           RETURNING
             AFP-RET-CODE.
  MOVE "CBLGDrawObjects" TO AFP-ERRDATA.
  PERFORM CHKSUCC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

#### GOCA Object Type

Specifies what type of objects are to be drawn. Valid values are:

##### FILLET\_OBJ (for C) or FILLET-0BJ (for COBOL)

Fillets are curves that are created by joining the points with conceptual straight lines, and then drawing a curve that is tangential to the first line at its start point, the last line at its end point, and intermediate lines at their center points. The thickness and type of line is determined from the Set Line Parms function call. See Figure 18 on page 275 for an example.

##### MARKER\_OBJ (for C) or MARKER-0BJ (for COBOL)

A marker is a symbol that is used to indicate a position. The particular symbol that is drawn is selected with the Set Marker Parms function call.

##### LINES\_OBJ (for C) or LINES-0BJ (for COBOL)

A series of straight lines drawn from the current position to the specified positions, given as pairs of actual (X,Y) coordinates in the GPS. The thickness and type of line is determined from the Set Line Parms function call.

**RLINES\_OBJ (for C) or RLINES-0BJ (for COBOL)**

A series of straight lines drawn from the current position to the specified positions, given as offsets relative to the last position. The thickness and type of line is determined from the Set Line Params function call.

**Count**

The number of coordinates that are supplied in the XY Pair Array. You must count both the X and Y coordinates. For example, if you want to specify three sets of (X,Y) coordinates, the count is six.

**XY Pair Array**

The address of an array containing the X and Y coordinates in the GPS of where the objects are to be drawn. Units are in 1440ths per inch.

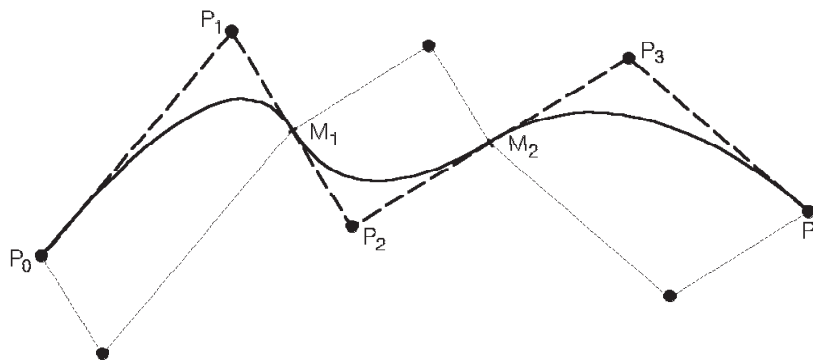


Figure 18. Fillets

Architecture Note: The Fillet drawing order does not support specification of a sharpness parameter. In Figure 9, this parameter would determine how close the drawn curve comes to the points  $P_1$ ,  $P_2$ , and  $P_3$ . If a quarter circle or quarter ellipse is used to fit the points, the sharpness parameter is not required since the circle or ellipse is completely defined by completing the parallelogram. If a quarter arc is not used, a sharpness parameter can be used and is defined, in reference to Figure 18, as follows:

1. Generate the virtual line  $P_0 M_1$ .
2. Find the midpoint of this line,  $V_0$ .
3. Generate  $V_0 P_1$ .
4. Call the point where  $V_0 P_1$  intersects the arc  $D_1$ .
5. The sharpness parameter is defined to be the ratio of  $V_0 D_1 \div D_1 P_1$ . The recommended value for the sharpness parameter, when used in AFP GOCA, is .7.

The current values of the line attributes are taken into account when the fillet is drawn. Current position is set to the last point specified.

The curve that is drawn is computed as follows (see Figure 18).

1. Let the points specified in the order be known as  $P_0, P_1, \dots, P_n$ .
2. Conceptual lines are drawn between the points  $P_0$  to  $P_1$ ,  $P_1$  to  $P_2$ ,  $P_2$  to  $P_3$ , and so on.
3. The midpoints of the lines from  $P_1$  to  $P_2$ ,  $P_2$  to  $P_3$ ,  $P_{n-2}$  to  $P_{n-1}$  are computed; call these  $M_1, M_2, \dots, M_{n-2}$ .
4. The points  $P_0, P_1, M_1, P_2, M_2, P_3, \dots, M_{n-2}, P_{n-1}, P_n$  are then considered three at a time, starting with  $P_0, P_1, M_1$ . A quadrant of a circle is scaled, and can be distorted to become a part of an ellipse, in order that the curve be tangential to the line  $P_0-P_1$  at the point  $P_0$ , and tangential to the line  $P_1-M_1$  at the point  $M_1$ .

The center point of the ellipse is the point obtained by completing the parallelogram defined by the sides  $P_0-P_1$  and  $P_1-M_1$ .

## GOCA Draw Objects

5. The next three points are considered, that is  $M_1$ ,  $P_2$ ,  $M_2$  and a quadrant of a circle is transformed into part of an ellipse that is tangential to the line  $M_1$ – $P_2$  at  $M_1$ , and tangential to the line  $P_2$ – $M_2$  at  $M_2$ .
6. This process continues, with part of an ellipse being fitted to three points in turn, until the last three points  $M_{n-2}$ ,  $P_{n-1}$ ,  $P_n$ , have been incorporated; see Figure 18 on page 275.

**Note:** If all the points  $P_0$  through  $P_n$  are within the GPS, the actual fillet does not go outside the GPS.

## Return Codes

none

## Sample Function Call

Draw a series of straight lines from the current position.

- **For the AFP Toolbox C Library:**

```
short rpscount;
short xypairs[20];
xypairs[0] = -100;
xypairs[1] = -100;
xypairs[2] = 0;
xypairs[3] = 0;
xypairs[4] = 100;
xypairs[5] = 100;
rc = AFPDrawObjects(GrHandle,LINES_OBJ,6,xypairs);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE 6 TO AFP-XYCOUNT.
MOVE LINES-OBJ TO AFP-GOCAOBJ-TYPE.
MOVE -2200 TO AFP-XYPEAIR-VALUE(1).
MOVE -2200 TO AFP-XYPEAIR-VALUE(2).
MOVE -1200 TO AFP-XYPEAIR-VALUE(3).
MOVE -1548 TO AFP-XYPEAIR-VALUE(4).
MOVE -200 TO AFP-XYPEAIR-VALUE(5).
MOVE 800 TO AFP-XYPEAIR-VALUE(6).
PERFORM AFPGOBJS.
```

## GOCA Ellipse

### Function

Draw an ellipse at the current GPS position. The ellipse is filled with a pattern if it is inside a graphics area and you have specified a fill pattern with GOCA Set Pattern Symbol.

This drawing order does not change the current GPS position. Different ellipses are shown in Figure 19 on page 278.

### Syntax

#### For the AFP Toolbox C Library:

```
AFPGEllipse (TBHANDLE      in_gocaHandle,
             short          in_xmaj,
             short          in_ymaj,
             short          in_xmin,
             short          in_ymin
            );
```

#### For the AFP Toolbox COBOL Interface:

```
AFPGEPLPS.
  CALL      "CBLGEllipse"
           USING
           BY VALUE
           AFP-GOCA-HANDLE
           AFP-XMAJOR
           AFP-YMAJOR
           AFP-XMINOR
           AFP-YMINOR
           RETURNING
           AFP-RET-CODE.
  MOVE "CBLGEllipse" TO AFP-ERRDATA.
  PERFORM CHKSUCC.
```

## GOCA Ellipse

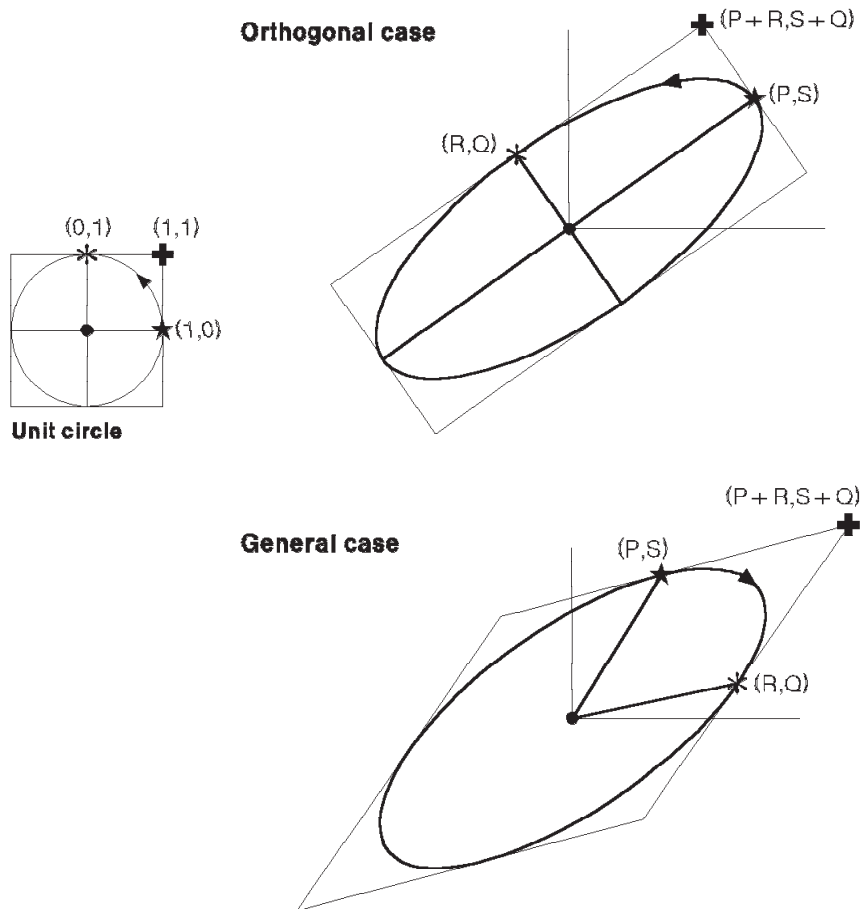


Figure 19. Ellipses

## Input Parameters

### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

### X and Y Major

These are the coordinates of the end point of the major axis of the ellipse. In Figure 19, P is the X major coordinate and S is the Y major coordinate.

### X and Y Minor

These are the coordinates of the end point of the minor axis of the ellipse. In Figure 19, R is the X minor coordinate and Q is the Y minor coordinate.

## Return Codes

none

## Sample Function Call

Draw an ellipse with its center at the current position, (X,Y) major at (100, 150), and (X,Y) minor at (-20, 25).

- **For the AFP C Toolbox Library:**

```
rc = AFPGEllipse(GrHandle,MM_2_U1440(100), MM_2_U1440(150)
                MM_2_U1440(-20), MM_2_U1440(25));
```

- **For the AFP COBOL Interface:**

```
MOVE 100 TO AFP-XMAJOR.  
MOVE 150 TO AFP-YMAJOR.  
MOVE -20 TO AFP-XMINOR.  
MOVE 25 TO AFP-YMINOR.  
PERFORM AFPGELPS.
```

## GOCA End Area

### Function

Ends an area inside a GOCA object. See Begin Area for more information.

This drawing order does not change the current GPS position.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPGEEndArea (TBHANDLE      in_gocaHandle,  
              short          in_padlength  
              );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGEA.  
    CALL "CBLGEEndArea"  
        USING  
        BY VALUE  
        AFP-GOCA-HANDLE  
        AFP-PADLENGTH  
        RETURNING  
        AFP-RET-CODE.  
    MOVE "CBLGEEndArea" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

#### Pad length

Number of bytes of zeros to use to pad this drawing order. Normally you should specify zero for this length.

### Return Codes

none

### Sample Function Call

See “GOCA Begin Area” on page 267 for an example.



---

## GOCA No Operation

### Function

Generates a single byte of zeros in the graphics object. This drawing order is ignored by the printer.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPGNOP (TBHANDLE      in_gocaHandle);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGNOP.  
    CALL "CBLGNOP"  
        USING  
        BY VALUE  
        AFP-GOCA-HANDLE  
        RETURNING  
        AFP-RET-CODE.  
    MOVE "CBLGNOP" TO AFP-ERRDATA.  
    PERFORM CHKSUC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

### Return Codes

none

### Sample Function Call

- **For the AFP Toolbox C Library:**

```
rc = AFPGNOP(GrHandle);
```

- **For the AFP Toolbox COBOL Interface:**

```
PERFORM AFPGNOP.
```

# GOCA Partial Circle

## Function

This function is used to draw part of a circle. It draws an arc and the radius on each end of the arc. See Figure 20 on page 283 for an example.

## Syntax

**For the AFP Toolbox C Library:**

```
AFPGPartCircle(  
    TBHANDLE in_gocaHandle,  
    short in_rad,  
    short in_xpos,  
    short in_ypos,  
    long in_start_ang,  
    long in_sweep_ang  
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGPCRC.  
    CALL      "CBLGPartCircle"  
            USING  
            BY VALUE  
            AFP-GOCA-HANDLE  
            AFP-RADIUS  
            AFP-ARCXPOS  
            AFP-ARCYPOS  
            AFP-ANGLE-START  
            AFP-ANGLE-SWEEP  
            RETURNING  
            AFP-RET-CODE.  
    MOVE "CBLGCircle" TO AFP-ERRDATA.  
    PERFORM CHKSUCC.
```

## Input Parameters

### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

### Radius

The distance from the center of the partial circle to the outside edge.

### ARC X and Y position

The center of the circle.

### Start angle

The angle, measured counterclockwise, between the X axis and the start point of the arc. This is illustrated in Figure 20 on page 283.

### Sweep angle

The angle, measured counterclockwise, between the two radii drawn from the center of the circle to the start and end points of the arc. This is illustrated in Figure 20 on page 283.

## Return Codes

None

## Sample Function Call

Draw a partial circle with radius 10, with the center at (50, 50), where the partial circle starts 30° from the X axis and continues for 60° beyond that.

- **For the AFP Toolbox C Library:**

```
rc = AFPGPartCircle(GrHandle, MM_2_U1440(10), 50, 50, 30, 60);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE 10 TO AFP-RADIUS  
MOVE 50 TO AFP-ARCXPOS  
MOVE 50 TO AFP-ARCYPOS  
MOVE 30 TO AFP-ANGLE-START  
MOVE 60 TO AFP-ANGLE-SWEEP  
PERFORM AFPGPCRC
```

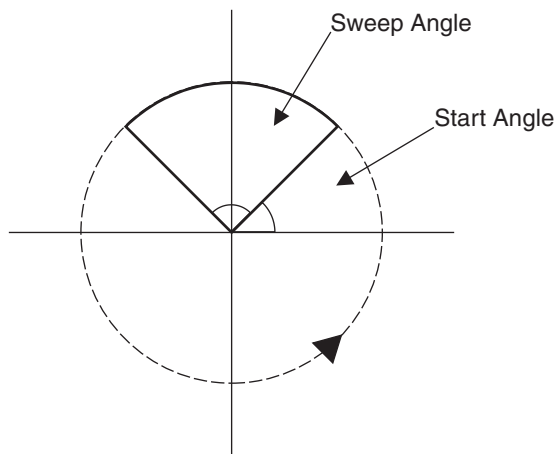


Figure 20. Partial Circle

# GOCA Partial Ellipse

## Function

This function is used to draw part of an ellipse. It draws an arc and the radius on each end of the arc. See Figure 21 on page 285 for an example.

## Syntax

**For the AFP Toolbox C Library:**

```
int AOCL_FUNCTION AFPGPartEllipse(  
TBHANDLE in_gocaHandle,  
                                short in_xmajor,  
short in_ymajor,  
short in_xminor,  
                                short in_yminor,  
short in_xpos,  
short in_ypos,  
                                long in_start_ang,  
long in_sweep_ang  
);
```

## Input Parameters

### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

### X and Y major

The coordinates of the end point of the major axis of the ellipse. In Figure 19 on page 278, P is the X major coordinate and S is the Y major coordinate.

### X and Y minor

The coordinates of the end point of the minor axis of the ellipse. In Figure 19 on page 278, R is the X minor coordinate and Q is the Y minor coordinate.

### X and Y position

The center of the ellipse.

### Start angle

The angle, measured counterclockwise, between the X axis and the start point of the arc. This is illustrated in Figure 21 on page 285.

### Sweep angle

The angle, measured counterclockwise, between the two radii drawn from the center of the circle to the start and end points of the arc. This is illustrated in Figure 21 on page 285.

## Return Codes

None

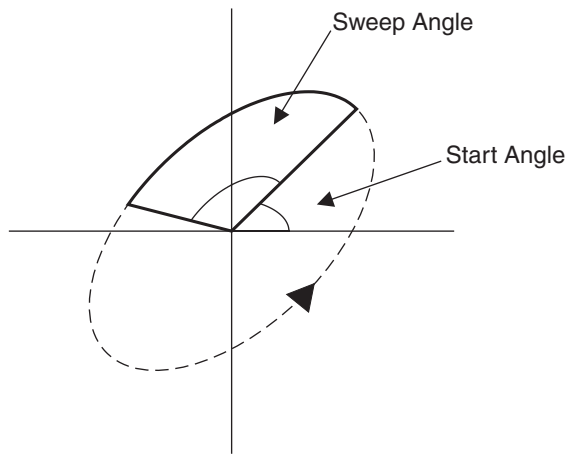


Figure 21. Partial Ellipse

# GOCA Query GPS Position

## Function

Queries the current GPS X and Y coordinates as determined by the last Set GPS Position or drawing order that caused the position to move.

## Syntax

**For the AFP Toolbox C Library:**

```
AFPGQueryGPSPos(TBHANDLE    in_gocaHandle,  
                short*       out_XPos,  
                short*       out_YPos  
                );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGQGPS.  
  CALL    "CBLGQueryGPSPos"  
        USING  
        BY VALUE  
        AFP-GOCA-HANDLE  
        BY REFERENCE  
        AFP-GPSXPOS  
        AFP-GPSYPOS  
        AFP-RET-CODE.  
  MOVE "CBLGQueryGPSPos" TO AFP-ERRDATA.  
  PERFORM CHKSUCC.
```

## Input Parameters

### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

### X and Y Position

Pointers to fields that contain the returned position information.

## Return Codes

none

## Sample Function Call

- **For the AFP Toolbox C Library:**

```
short xpos, ypos;  
rc = AFPGQueryGPSPos(GrHandle,&xpos,&ypos);
```

- **For the AFP Toolbox COBOL Interface:**

```
PERFORM AFPGQGPS.  
  DISPLAY "XPOS AFTER QUERY " AFP-GPSXPOS.  
  DISPLAY "YPOS AFTER QUERY " AFP-GPSYPOS.
```

## GOCA Set Character Parameters

### Function

Sets the values to be used in drawing text strings within the graphic object.

**Note:** Some older printers do not handle GOCA text correctly. If the text does not format as you think it should, place the character string into your page using AFP Write instead of placing it inside the GOCA object.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPGSetCharParms(TBHANDLE      in_gocaHandle,
                  short          in_anglexpos,
                  short          in_angleypos,
                  short          in_cellwidth,
                  short          in_cellheight,
                  short          in_cellwidth_fr,
                  short          in_cellheight_fr,
                  CHARACTERDIR    in_chardirection,
                  PRECISION      in_charprecision
                  );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGSCP.
    CALL "CBLGSetCharParms"
        USING
            BY VALUE
                AFP-GOCA-HANDLE
                AFP-ANGLE-XPOS
                AFP-ANGLE-YPOS
                AFP-CELLWIDTH
                AFP-CELLHEIGHT
                AFP-CELLWFR
                AFP-CELLHFR
                AFP-CHARDIR
                AFP-PRECISION
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLGSetCharParms" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

## Input Parameters

### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

### Angle X and Y Positions

The character angle controls the angle of the character baseline with respect to the GPS X axis, measured counter-clockwise. This is similar to the character rotation function in presentation text (Set Text Orientation). In AFP printers only angles of 0, 90, 180, and 270 degrees are supported. Select the angle you want by setting X and Y appropriately:

If X position is positive and Y position is zero, the character angle is zero degrees.

If X position is zero and Y position is positive, the character angle is 90 degrees.

If X position is negative and Y position is zero, the character angle is 180 degrees.

If X position is zero and Y position is negative, the character angle is 270 degrees.

## GOCA Set Character ParmS

### Cell width and height

These four parameters set the value of the current character cell-size, used to scale characters specified in subsequent Graphics Character String drawing orders. You can specify both an integer and fractional portion for the width and height values.

**Note:** You might achieve better results by using an outline font and scaling it on the Define Font command.

### Character Direction

This parameter controls the placement of a character relative to the previous character. Valid values are:

**DEFAULT\_CD (for C) or DEFAULT-CD (for COBOL)**

Uses the drawing default, which in AFP environments is the same as LFT2RGHT.

**LFT2RGHT\_CD (for C) or LFT2RGHT-CD (for COBOL)**

Positions character boxes from left to right along the GPS X axis.

**TOP2BTM\_CD (for C) or TOP2BTM-CD (for COBOL)**

Positions character boxes from top to bottom along the GPS X axis.

**RGHT2LFT\_CD (for C) or RGHT2LFT-CD (for COBOL)**

Positions character boxes from right to left along the GPS X axis.

**BTM2TOP\_CD (for C) or BTM2TOP-CD (for COBOL)**

Positions character boxes from bottom to top along the GPS X axis.

Figure 22 on page 289 shows examples of character angle and direction.

### Character Precision

This parameter specifies how the characters are to be drawn.

**DEFAULT\_PR (for C) or DEFAULT-PR (for COBOL)**

Uses the drawing default. In AFP printers this is the same as CHARACTER precision.

**STRING\_PR (for C) or STRING-PR (for COBOL)**

Draws the characters using the simplest mode possible for the implementation. Some character attributes might be ignored as a trade-off for improved performance.

**CHARACTER\_PR (for C) or CHARACTER-PR (for COBOL)**

Draws the characters taking into account all the attributes to determine the positioning of the characters.

## Return Codes

none



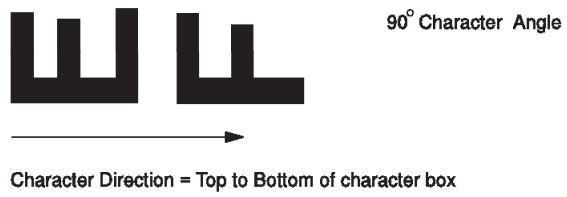
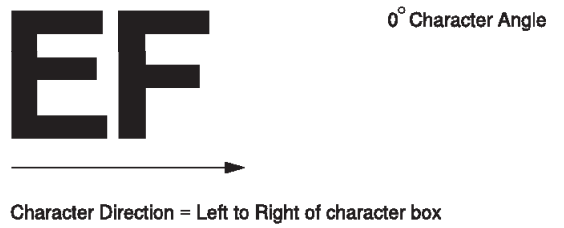


Figure 22. Character Angles and Directions

# GOCA Set Font

## Function

Set the current font for GOCA text only. This command does not effect text in the page.

## Syntax

### For the AFP Toolbox C Library:

```
AFPGSetFont(TBHANDLE      in_gocaHandle,
            FontID         in_fontid
            );
```

### For the AFP Toolbox COBOL Interface:

```
AFPGSFNT.
  CALL    "CBLGSetFont"
          USING
          BY VALUE
          AFP-GOCA-HANDLE
          AFP-FONT-ID
          RETURNING
          AFP-RET-CODE.
  MOVE "CBLGSetFont" TO AFP-ERRDATA.
  PERFORM CHKSUCC.
```

## Input Parameters

### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

### Font ID

A font ID returned by any of the normal AFP Define Font commands. Specify zero for the fontID to request that the default GOCA font for the device should be used.

## Return Codes

none

## Sample Function Call

Set the current GPS font to Helvetica 10 point:

- **For the AFP Toolbox C Library:**

```
rc = AFPDefineFontByAttr ("T1V10500","HELVETICA LATIN1",100,
                        MEDIUM_WT,MEDIUM_WD,NORMAL_FS,&Helv10);
rc = AFPGSetFont(GrHandle,Helv10);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE "T1V10500" TO AFP-CODE-PAGE.
MOVE "HELVETICA LATIN1"
  TO AFP-DESCRIPTIVE-NAME.
MOVE 100 TO AFP-POINT-SIZE.
MOVE MEDIUM TO AFP-WEIGHT.
MOVE NORML TO AFP-FONT-WIDTH.
MOVE ROMAN TO AFP-STYLE.
PERFORM AFPDFNTAT.
PERFORM AFPGSFNT.
```

## GOCA Set GPS Position

### Function

Set the current X and Y coordinates in the Graphic Presentation Space. This command has no effect on the current page position (Set Position).

### Syntax

**For the AFP Toolbox C Library:**

```
AFPGSetGPSPos(TBHANDLE      in_gocaHandle,
               short          in_XPos,
               short          in_YPos
               );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGSGPS.
  CALL      "CBLGSetGPSPos"
           USING
           BY VALUE
             AFP-GOCA-HANDLE
             AFP-GPSXPOS
             AFP-GPSYPOS
           BY REFERENCE
             AFP-RET-CODE.
  MOVE "CBLGSetGPSPos" TO AFP-ERRDATA.
  PERFORM CHKSUCC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

#### X and Y Positions

Pointers to fields that will contain the returned position information.

### Return Codes

none

### Sample Function Call

Set the current GPS position to X = 50mm, Y = -25mm.

- For the AFP Toolbox C Library:**

```
rc = AFPGSetGPSPos(GrHandle,MM_2_U1440(50),MM_2_U1440(-25));
```

- For the AFP Toolbox COBOL Interface:**

```
MOVE "MM" TO MY-UOM.
MOVE 50 TO MY-ORIGINAL-VALUE.
CALL "ATXUNITS" USING
  MY-UOM, MY-ORIGINAL-VALUE,
  MY-NEW-VALUE.
MOVE MY-NEW-VALUE TO AFP-GPSXPOS.
MOVE -25 TO MY-ORIGINAL-VALUE.
CALL "ATXUNITS" USING
  MY-UOM, MY-ORIGINAL-VALUE,
  MY-NEW-VALUE.
MOVE MY-NEW-VALUE TO AFP-GPSYPOS.
PERFORM AFPGSGPS.
```

## GOCA Set Extended Color

---

### Function

Set the foreground color attributes for characters, image, lines, markers, and patterns.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPGSetColor(
    TBHANDLE      in_gocaHandle,
    MODCAColors   in_color
);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGSEXC.
    CALL "CBLGSetColor"
        USING
            BY VALUE
                AFP-GOCA-HANDLE
                AFP-COLOR
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLGSetColor" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

#### Color

Any of the defined MO:DCA color values. Valid values are:

**BLUE (for C, COBOL, or RPG)**

**RED (for C, COBOL, or RPG)**

**MAGENTA or PINK (for C, COBOL, or RPG)**

**GREEN (for C, COBOL, or RPG)**

**CYAN or TURQ (for C, COBOL, or RPG)**

**YELLOW (for C, COBOL, or RPG)**

**BLACK (for C, COBOL, or RPG)**

**BROWN (for C, COBOL, or RPG)**

**MUSTARD (for C, COBOL, or RPG)**

**DARKBLUE (for C, COBOL, or RPG)**

**DARKGREEN (for C, COBOL, or RPG)**

**DARKTURQ or DARKCYAN (for C, COBOL, or RPG)**

**ORANGE (for C, COBOL, or RPG)**

**PURPLE (for C, COBOL, or RPG)**

**GRAY (for C, COBOL, or RPG)**

NONE (for C, COBOL, or RPG)

MEDIUM\_COLOR (for C) or MEDIUM-COLOR (for COBOL)

DEFAULT\_COLOR (for C) or DEFAULT-COLOR (for RPG)

## Return Codes

none

## Sample Function Call

Set the color for subsequent items to blue:

- **For the AFP Toolbox C Library:**

```
rc = AFPGSetColor(GrHandle,BLUE);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE BLUE TO AFP-COLOR.  
PERFORM AFPGSEXC.
```

## GOCA Set Line Parameters

---

### Function

Set the width and type of line to be drawn with subsequent Draw Box, Circle, Ellipse, and Draw Object commands.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPGSetLineParms(TBHANDLE      in_gocaHandle,
                  LINETYPE      in_linetype,
                  byte           in_linewidth
                  );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGSLPR.
    CALL "CBLGSetLineParms"
        USING
        BY VALUE
            AFP-GOCA-HANDLE
            AFP-LINETYPE
            AFP-LINEWIDTH
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLGSetLineParms" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

#### Line Type

Specifies the pattern to be used for drawing lines:

**For the AFP Toolbox C Library:**

**DEFAULT\_LT (for C) or DEFAULT-LT (for COBOL)**

A solid line.

**DOTTED\_LT (for C) or DOTTED-LT (for COBOL)**

A dotted line.

**SHORTDSH\_LT (for C) or SHORTDSH-LT (for COBOL)**

A short dashed line.

**DASHDOT\_LT (for C) or DASHDOT-LT (for COBOL)**

A dash-dot line.

**DBLDOT\_LT (for C) or DBLDOT-LT (for COBOL)**

A double dotted line.

**LONGDSH\_LT (for C) or LONGDSH-LT (for COBOL)**

A long dashed line.

**DSHDBDOT\_LT (for C) or DSHDBDOT-LT (for COBOL)**

A dash-double-dot line.

**SOLID\_LT (for C) or SOLID-LT (for COBOL)**

A solid line.

**INVSBLT (for C) or INVSBLT-LT (for COBOL)**

An invisible line.

**Line Width**

Width as an integer multiplier of the normal line width (which is approximately .01 inches).

**Return Codes**

none

**Sample Function Call**

Specify that lines should be drawn with short dashes, twice as wide as normal.

- **For the AFP Toolbox C Library:**

```
rc = AFPSetLineParms(GrHandle,SHORTDASH_LT,2);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE SHORTDASH-LT TO AFP-LINETYPE  
MOVE 2 TO AFP-LINEWIDTH  
PERFORM AFPSETLPR.
```

## GOCA Set Marker Parameters

---

### Function

Set the symbol and precision to be used for drawing markers with the Draw Object command.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPGSetMarkerParams(TBHANDLE    in_gocaHandle,
                    PRECISION    in_precision,
                    MARKERSYM    in_symcode
                    );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGSMPR.
    CALL "CBLGSetMarkerParams"
        USING
            BY VALUE
                AFP-GOCA-HANDLE
                AFP-MARKERSYMBOL
                AFP-PRECISION
            RETURNING
                AFP-RET-CODE.
    MOVE "CBLGSetMarkerParams" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

#### Marker Precision

This parameter specifies how precise the markers are to be placed.

##### **DEFAULT\_PR (for C) or DEFAULT-PR (for COBOL)**

Uses the drawing default. In AFP printers this is the same as CHARACTER precision.

##### **STRING\_PR (for C) or STRING-PR (for COBOL)**

Draws the markers using the simplest mode possible for the implementation. Positioning might be approximate.

##### **CHARACTER\_PR (for C) or CHARACTER-PR (for COBOL)**

Draws the markers taking into account all the attributes to determine the positioning of the markers.

#### Marker Symbol

Specifies the symbol to be drawn as a marker:

##### **DEFAULT\_MS (for C) or DEFAULT-MS (for COBOL)**

The same as a cross.

##### **CROSS\_MS (for C) or CROSS-MS (for COBOL)**

Draws an X.

##### **PLUS\_MS (for C) or PLUS-MS (for COBOL)**

Draws a plus sign.

##### **DIAMOND\_MS (for C) or DIAMOND-MS (for COBOL)**

Draws a hollow diamond.



**SQUARE\_MS (for C) or SQUARE-MS (for COBOL)**

Draws a hollow square.

**SIXSTAR\_MS (for C) or SIXSTAR-MS (for COBOL)**

Draws a 6-point star.

**EIGHTSTR\_MS (for C) or EIGHTSTAR-MS (for COBOL)**

Draws a 8-point star.

**FILLDMND\_MS (for C) or FILLDMND-MS (for COBOL)**

Draws a filled diamond.

**FILLSQR\_MS (for C) or FILLSQR-MS (for COBOL)**

Draws a filled square.

**DOT\_MS (for C) or DOT-MS (for COBOL)**

Draws a solid dot.

**CIRCLE\_MS (for C) or CIRCLE-MS (for COBOL)**

Draws a circle.

## Return Codes

none

## Sample Function Call

Specify that eight pointed stars are used as marker symbols.

- **For the AFP Toolbox C Library:**

```
rc = AFPGSetMarkerParams(GrHandle,STRING_PR,EIGHTSTR_MS);
```

- **For the AFP Toolbox C Library:**

```
MOVE EIGHTSTR-MS TO AFP-MARKERSYMBOL
MOVE STRING-PR TO AFP-PRECISION
PERFORM AFPGSMR.
```

---

# GOCA Set Pattern Symbol

## Function

Specifies the type of fill pattern to use for boxes, circles, ellipses, and other paths that are drawn inside a graphic area.

## Syntax

**For the AFP Toolbox C Library:**

```
AFPGSetPatternSym(TBHANDLE    in_gocaHandle,  
                  FILLPATTERN in_pattern);
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGSPSM.  
  CALL    "CBLGSetPatternSym"  
        USING  
        BY VALUE  
        AFP-GOCA-HANDLE  
        AFP-PATTERN  
        RETURNING  
        AFP-RET-CODE.  
  MOVE "CBLGSetPatternSym" TO AFP-ERRDATA.  
  PERFORM CHKSUCC.
```

## Input Parameters

### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

### Pattern Symbol

Specifies the type of fill pattern to use. See Figure 23 on page 299 for samples.

#### **DEFAULT\_FP (for C), DEFAULT-PR (for COBOL)**

Solid fill. See fill pattern 16 in the figure.

#### **D0T01\_FP (for C), D0T01-FP through D0T08-FP or D0T08-FP (for COBOL)**

Dotted patterns of decreasing density. See fill patterns 1-8 in the figure.

#### **VERTLN\_FP (for C), VERTLN-FP (for COBOL)**

Vertical lines. See fill pattern 9 in the figure.

#### **H0RZLN\_FP (for C), H0RZLN-FP (for COBOL)**

Horizontal lines. See fill pattern 10 in the figure.

#### **BLTR1\_FP and BLTR2\_FP (for C), BLTR-FP and BLTR-FP (for COBOL)**

Diagonal lines from bottom-left to top-right. See fill patterns 11-12 in the figure.

#### **TLBR1\_FP and TLBR2\_FP (for C), TLBR1-FP and TLBR2-FP (for COBOL)**

Diagonal lines from top-left to bottom-right. See fill patterns 13-14 in the figure.

#### **N0FILL\_FP (for C), N0FILL-FP (for COBOL)**

No fill pattern. See fill pattern 15 in the figure.

#### **S0LID\_FP (for C), S0LID-FP (for COBOL)**

Solid fill. See fill pattern 16 in the figure.

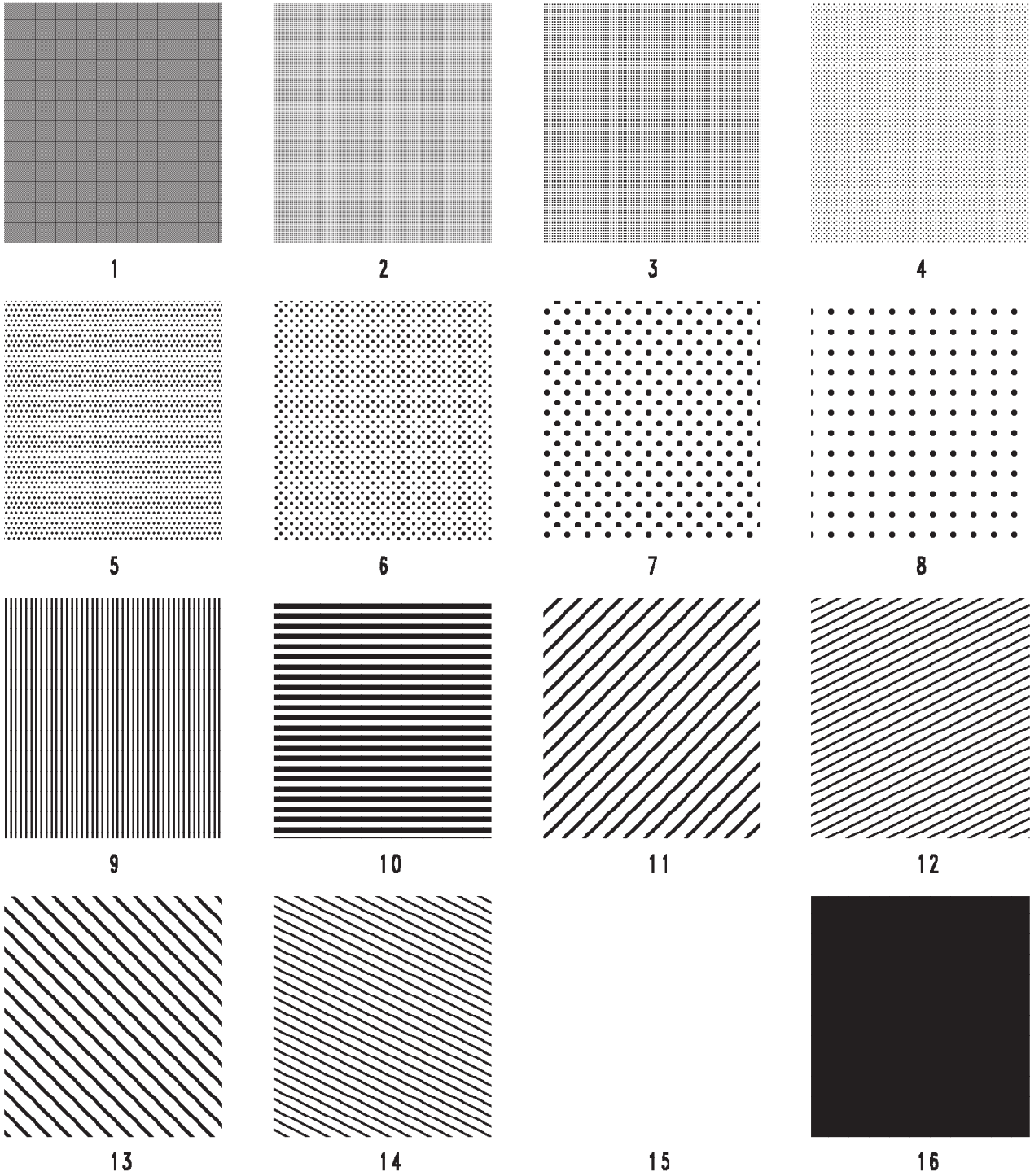


Figure 23. Fill Patterns

Return Codes

none

## GOCA Set Pattern Symbol

### Sample Function Call

Specify that filled areas should be done with vertical lines.

- **For the AFP Toolbox C Library:**

```
rc = AFPGSetPatternSym(GrHandle,VERTLN_FP);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE VERTLN-FP TO AFP-PATTERN  
PERFORM AFPGSPSM.
```

## GOCA Set Process Color

### Function

Specifies a process color, highlight color, or named color to be used for characters, images, lines, markers, and patterns.

**Note:** Use of this drawing order requires microcode support in your printer.

### Syntax

**For the AFP Toolbox C Library:**

```
AFPGSetProcessColor(TBHANDLE in_gocaHandle,
                    ColorSpace in_coltype,
                    MODCAColors in_color,
                    ulong in_colspec1,
                    ulong in_colspec2,
                    ulong in_colspec3,
                    ulong in_colspec4
                    );
```

**For the AFP Toolbox COBOL Interface:**

```
AFPGSPCL.
    CALL "CBLGSetProcessColor"
        USING
        BY VALUE
            AFP-GOCA-HANDLE,
            AFP-COLOR-TYPE,
            AFP-COLOR,
            AFP-COLSPEC1,
            AFP-COLSPEC2,
            AFP-COLSPEC3,
            AFP-COLSPEC4,
        RETURNING
            AFP-RET-CODE.
    MOVE "CBLGSetProcessColor" TO AFP-ERRDATA.
    PERFORM CHKSUCC.
```

### Input Parameters

#### GOCA Handle

The handle for the GOCA object to contain this drawing order. GOCA handle is returned by the Begin Graphic call.

#### Color Space Type

Specifies the type of color to be generated:

##### RGB\_COLOR (for C) or RGB-COLOR (for COBOL)

The color value is specified as three intensity levels for the red, green, and blue components of the color. These numbers are given for color specification values 1, 2, and 3. The fourth specification is ignored and should be specified as 0.

##### CMYK\_COLOR (for C) or CMYK-COLOR (for COBOL)

The color value is specified as four intensity levels for the cyan, magenta, yellow, and black components of the color. These are color specifications 1-4.

##### HILIGHT\_COLOR (for C) or HILIGHT-COLOR (for COBOL)

Defines a request for presentation device to generate a highlight color. Specify a MO:DCA Color value for the highlight color. Color specification 1 is the percentage coverage for the

## GOCA Set Process Color

specified color, and color specification 2 is the percentage of black that is to be added to the coverage (can be 0). Color specifications 3 and 4 should be 0.

### **CIELAB\_COLOR (for C) or CIELAB-COLOR (for COBOL)**

Specified with three components. Color specification 1 is the luminance, and color specifications 2 and 3 are the chrominance differences. Color specification 4 should be 0.

### **OCA\_COLOR (for C) or OCA-COLOR (for COBOL)**

Defines a request for presentation device to generate a highlight color. Specify a MO:DCA Color value for the highlight color. The presentation device will produce printed colors using a color mapping table (see PSF documentation for your environment). Color specifications 1-4 should be 0.

## MODCA Color

### Color

Any of the defined MO:DCA color values. Valid values are:

**BLUE (for C, COBOL, or RPG)**

**RED (for C, COBOL, or RPG)**

**MAGENTA or PINK (for C, COBOL, or RPG)**

**GREEN (for C, COBOL, or RPG)**

**CYAN or TURQ (for C, COBOL, or RPG)**

**YELLOW (for C, COBOL, or RPG)**

**BLACK (for C, COBOL, or RPG)**

**BROWN (for C, COBOL, or RPG)**

**MUSTARD (for C, COBOL, or RPG)**

**DARKBLUE (for C, COBOL, or RPG)**

**DARKGREEN (for C, COBOL, or RPG)**

**DARKTURQ or DARKCYAN (for C, COBOL, or RPG)**

**ORANGE (for C, COBOL, or RPG)**

**PURPLE (for C, COBOL, or RPG)**

**GRAY (for C, COBOL, or RPG)**

**NONE (for C, COBOL, or RPG)**

**MEDIUM\_COLOR (for C) or MEDIUM-COLOR (for COBOL)**

**DEFAULT\_COLOR (for C) or DEFAULT-COLOR (for RPG)**

### Color specifications 1-4

Usage depends on the color space type, see descriptions above.

## Return Codes

none

## Sample Function Call

Specify that color output should be a blue highlight color space, with 75 percent blue coverage and 25 percent black.

- **For the AFP Toolbox C Library:**

```
rc = AFPGSetProcessColor(GrHandle,HILIGHT_CS,BLUE,75,25,0,0);
```

- **For the AFP Toolbox COBOL Interface:**

```
MOVE HILIGHT-CS TO AFP-COLOR-TYPE.  
MOVE 75 TO AFP-COLSPEC1.  
MOVE 25 TO AFP-COLSPEC2.  
MOVE 0 TO AFP-COLSPEC3.  
MOVE 0 TO AFP-COLSPEC4.  
MOVE BLUE TO AFP-COLOR.  
PERFORM AFPDSPCL.
```

## GOCA Set Process Color



---

## Part 3. AFP Toolbox for C++ Language

<b>Chapter 5. AFP Toolbox C++ Object Library</b>	309
Using Objects to Build an AFP Document	309
AEString	310
Including C++ Header Files	311
Document.	312
Declarations for the Document Class	312
Constructors and Operators	312
Public Members of Document	312
DocName	312
Mnemonic.	312
NumPages	313
OutputThreshold	313
AFPDoc	314
Declarations for the AFPDoc Class	314
SetDocumentCodepage.	314
SetInputCodepage	314
SetInputDbCodeset	315
Constructors and Operators	315
Public Members of AFPDoc	315
AddPage	315
BeginGroup	315
Comment	316
EndGroup.	316
InvokeMediumMap	316
Link	316
Mnemonic.	316
PreloadObject	316
PreloadOverlay	317
Tag	317
TimeStamp	317
Page Class	318
Declarations for the Page Class.	318
Constructors and Operators	318
Public Members of Page	318
CopyPage.	318
IsValid	318
Mnemonic.	318
AFPPage	319
Declarations for the AFPPage Class	319
Constructors and Operators	320
Public Members of AFPPage.	320
Comment	320
CopyPage.	320
IncludeBarCode	320
IncludeBarCodeInline	320
IncludeGraphic	321
IncludeGraphicInline	321
IncludeImage	321
IncludeImageInline	321
IncludeObject	321
IncludeOverlay	323
IncludePageSegment	323
IncludeText	324

KeepObject . . . . .	324
Link . . . . .	324
Mnemonic . . . . .	324
PageName . . . . .	324
PageSize . . . . .	324
SetObjectColorProfile . . . . .	325
Tag . . . . .	325
RotateOverlay . . . . .	325
ShadeBlock . . . . .	325
PText . . . . .	326
Declarations for the PText Class . . . . .	326
Constructors and Operators . . . . .	327
Public Members of PText . . . . .	327
AddText . . . . .	327
AddDbText . . . . .	328
Color . . . . .	328
DrawBox . . . . .	329
DrawRule . . . . .	329
Hmove . . . . .	329
HMoveTo . . . . .	329
MaxPTXLength . . . . .	329
Mnemonic . . . . .	330
Move . . . . .	330
MoveTo . . . . .	330
RepeatString . . . . .	330
SetFont . . . . .	330
SetTextOrientation . . . . .	330
Vmove . . . . .	331
VMoveTo . . . . .	332
OCA . . . . .	333
Declarations for the OCA Class . . . . .	333
Constructors and Operators . . . . .	334
Public Members of OCA . . . . .	334
CopyOCA . . . . .	334
Mnemonic . . . . .	334
ObjectAreaContentPosition . . . . .	334
ObjectAreaMeasurementUnits . . . . .	334
ObjectAreaMixingFlag . . . . .	334
ObjectAreaPosition . . . . .	335
ObjectAreaSize . . . . .	335
ObjectContentMapping . . . . .	335
OCAType . . . . .	336
Image Object Content Architecture (IOCA) . . . . .	337
Declarations for the IOCA Class . . . . .	337
Constructors and Operators . . . . .	338
Public Members of IOCA . . . . .	339
BeginImage . . . . .	339
CopyOca . . . . .	339
EndImage . . . . .	339
Export . . . . .	339
IDESize . . . . .	339
IDestructure . . . . .	339
ImageData . . . . .	340
ImageEncoding . . . . .	340
LookUpTable . . . . .	340
MaxIPDLength . . . . .	340

Mnemonic. . . . .	340
Bar Code Object Content Architecture (BCOCA). . . . .	342
Declarations for the BCOCA Class. . . . .	342
Constructors and Operators . . . . .	342
Public Members of BCOCA . . . . .	344
AddBarCode . . . . .	344
CopyOca . . . . .	345
Export . . . . .	345
Font . . . . .	345
Mnemonic. . . . .	345
Graphic Object Content Architecture (GOCA). . . . .	346
Declarations for the GOCA Class . . . . .	346
Constructors and Operators . . . . .	347
Public Members of GOCA . . . . .	347
BeginArea. . . . .	347
CharString . . . . .	348
DrawBox . . . . .	348
DrawFillet . . . . .	348
DrawMarkers . . . . .	349
DrawRules . . . . .	349
EndArea . . . . .	349
FullArc . . . . .	349
NoOP . . . . .	349
QueryGPSPos . . . . .	349
RelativeRules . . . . .	349
SetArcParms. . . . .	350
SetCharAngle . . . . .	350
SetCharCell . . . . .	350
SetCharDir . . . . .	350
SetCharPrec . . . . .	350
SetCharSet . . . . .	351
SetGPSPos . . . . .	351
SetExtColor . . . . .	351
SetLineType . . . . .	351
SetLineWidth . . . . .	351
SetMarkerPrec . . . . .	351
SetMarkerSym . . . . .	351
SetPatternSym . . . . .	352
SetProcessColor . . . . .	352
<b>Chapter 6. Defining Fonts with C++ Language Programs . . . . .</b>	<b>355</b>
Define Double-Byte Font By Attribute. . . . .	356
Define Font By Attribute . . . . .	358
Define Font By Attribute With Scaling. . . . .	360
Define Font By Name . . . . .	362
Measure Double-Byte String . . . . .	363
Measure String . . . . .	365
Set Font Type . . . . .	367
Translate And Measure Double-Byte String . . . . .	368
Translate And Measure String . . . . .	370



---

## Chapter 5. AFP Toolbox C++ Object Library

This chapter describes the objects provided with AFP Toolbox for use by C++ language programs. To use these objects, you need to be aware of the structure of AFP documents, but do not need to understand the semantics or syntax of the MO:DCA data stream.

There are nine classes supported in the MO:DCA data stream. A description of each class can be found in the following sections:

### The Document Class

See “Document” on page 312.

### The AFPDoc Class

See “AFPDoc” on page 314.

### The Page Class

See “Page Class” on page 318.

### The AFPPage Class

See “AFPPage” on page 319.

### The Presentation Text (PText) Class

See “PText” on page 326.

### The Object Content Architecture (OCA) Class

See “OCA” on page 333.

### The Image Object Content Architecture (IOCA) Class

See “Image Object Content Architecture (IOCA)” on page 337.

### The Bar Code Object Content Architecture (BCOCA) Class

See “Bar Code Object Content Architecture (BCOCA)” on page 342.

### The Graphic Object Content Architecture (GOCA) Class

See “Graphic Object Content Architecture (GOCA)” on page 346.

The following topics are described for each of the classes in AFP Toolbox:

- Declarations for object classes
- Constructors and operators
- Public members

---

## Using Objects to Build an AFP Document

The normal sequence of events for an application using the AFPDoc C++ objects is:

1. Define the fonts used in this document using the AFPDefineFontByAttr, AFPDefineDbFontByAttr, AFPDefineFontByAttrWithScaling, and AFPDefineFontByName functions.
2. Create an AFPDoc object.
3. Create an AFPPage object.
  - a. Create a single Presentation Text (PText ) text object to contain all the text for this page.
    - 1) Specify where text data, rules, or boxes are put in the object using the Move, MoveTo, HMove, HMoveTo, VMove, and VMoveTo functions.
    - 2) Specify any data attributes using SetFont, Color, and SetTextOrientation functions.

**Note:** You only need to specify data attributes if they are different from those used for the previous data.

- 3) Place the data in the text object using DrawRule, RepeatString, AddText, AddDbText, or other text object functions. AddDbText must be used with double-byte character set (DBCS) fonts obtained by AFPDefineDbFontByAttr. You should not use other functions, such as RepeatString and AddText, with DBCS fonts.

## The C++ Object Library

- 4) Repeat steps 3a1 on page 309 through 3a3 on page 309 until you have placed all the text for the page.
  - b. Put the text object in the page using the IncludeText function of the AFPPage object. Delete the text object.
  - c. Put other objects in the page using the IncludeImage, IncludeGraphicInline, IncludeImageInline, IncludeObject, IncludeOverlay, IncludePageSegment, IncludeBarCode, Link, Comment, and Tag functions of the AFPPage object.
  - d. Repeat step 3c until the page is finished.
4. If this is the first page of a group of pages, call the BeginGroup function of the AFPDoc object.
5. Call the AddPage function of the AFPDoc object to put the page in the document. Delete the page object.
6. If this is the last page of a group of pages, call the EndGroup function of the AFPDoc object.
7. Repeat steps 3 on page 309 through 6 until all pages are finished.
8. Delete the document object.

---

## AEStrng

In order to manipulate text data back and forth between ASCII and EBCDIC code pages, the AEStrng class is used in place of the more typical **char\*** parameter. However, this parameter type is largely ignored because any place an AEStrng is used as an input parameter, a string constant, or char\* variable can be used. Any time an AEStrng is provided as output, the contents can be accessed with the toChar() function. For example, the statement:

```
AEStrng aestrng("Hello World");  
char* cptr=aestrng.toChar();
```

sets the **cptr** variable to point to the first character of the string "Hello World" represented in AEStrng.

The input code page for all text in AFP Toolbox is, by default, assumed to be in the multinational ASCII code page T1000850 on your workstation or EBCDIC code page T1V10500 on your host system. However, this code page can be overridden for all strings in a program by calling the

```
AEStrng::defaultCodepage(char* cpname, char defaultFillChar)
```

function, or on any individual AEStrng object, by specifying the code page as a constructor parameter. For example, both of the following two code segments create a string with the default code page of 850, and then a second string with a code page of 437. The first example changes the default for all AEStrings and the second just overrides it for one case, leaving the default as code page 850:

### Example 1

```
AEStrng str850("Codepage 850 string");  
AEStrng::defaultCodepage("T1000437");  
AEStrng str437("Codepage 437 string");
```

### Example 2

```
AEStrng str850("Codepage 850 string");  
AEStrng str437("Codepage 437 string","T1000437");
```

Programs can also translate AEStrng objects from code page to code page by using the **translate** function. For example:

```
AEStrng str850(" Created in default CP 850");  
str850.translate("T1V10500"); // translate to EBCDIC cp 500
```

When working with multi-byte character set (MBCS) character strings (also called double-byte character sets or DBCS), the input code set must be set in order to translate AEStrng objects from code set to code set by using the Dbtranslate function.

```
AEStrng mbcs("Created in code set of IBM-932");  
mbcs.Dbtranslate("IBM-939");
```

## Including C++ Header Files

In order to use the C++ functions described in this chapter, you must include the AFP Toolbox header file once in your application program:

```
#include "afptk.h"
```

The afptk.h file is supplied with the AFP Toolbox and installed by default in the following locations:

### **AIX System**

In the **/toolbox/samples** directory.

### **MVS System**

In **ATX.SATXSAMP(ATXAFPTK)** with the alias of AFPTK.

Contact your system administrator if you cannot find the afptk.h file.

**Note:** C++ is not supported on OS/400.

### Document

This section describes the members of the Document class and the operations that are common to all classes that are derived from Document. Document is an abstract class that serves as the base class to all types of document objects in AFP Toolbox.

### Declarations for the Document Class

To use the Document class functions, specify the following in your program:

```
#define INCL_DOCUMENT
#include "afptk.h"
```

The above statements cause AFP Toolbox to include the appropriate header files at compile time. The following is an excerpt from the **document.h** file that shows the relevant declarations for the members of Document.

```
class Document
{
public:
    Document(const Document&);
    virtual ~Document();
    Document& operator=(const Document&);
    virtual boolean IsValid(void) const;
    virtual AESTring DocName(void) const;
    AESTring Mnemonic() const;
    ulong NumPages(void) const;
    ulong OutputThreshold(void) const;
    void OutputThreshold(ulong newThreshold);
protected:
    Document(const AESTring& docName = "");
};
```

### Constructors and Operators

public:

```
    Document(const Document&);
    Document& operator=(const Document&);
```

protected:

```
    Document(const AESTring& docName = "");
```

The basic constructor takes a single argument which is the document name. However, this constructor is protected and is intended for use only by subclasses of Document, such as AFPDoc.

The assignment operator is used to copy Document objects.

### Public Members of Document

#### DocName

```
virtual AESTring DocName(void) const;
```

Queries the document name.

#### Mnemonic

```
AESTring Mnemonic() const;
```

Returns a mnemonic label ("Abstract Document") for this object to identify this object class.



**NumPages**

```
ulong NumPages(void) const;
```

Returns the number of pages currently in the Document object.

**OutputThreshold**

```
ulong OutputThreshold(void) const;  
void OutputThreshold(ulong newThreshold);
```

Sets or queries the number of pages to store in memory before writing them to disk. This function is not currently used in the AFP Toolbox.

### AFPDoc

The AFPDoc class is derived from the Document class and supports the creation of AFP documents. This section describes how to use AFPDoc.

### Declarations for the AFPDoc Class

To use the AFPDoc class functions, specify the following in your program. These statements cause AFP Toolbox to include the appropriate header files at compile time:

```
#define INCL_DOCUMENT
#include <afptk.h>
```

The following is an excerpt from the **document.h** file that shows the relevant declarations for the members of AFPDoc.

```
class AFPDoc: public Document
{
public:
    static void SetDocumentCodepage(const char* codepage, char spaceChar);
    static void SetInputCodepage(const char* codepage, char spaceChar);
    static void SetInputDbCodeset(const char* codeset);

    AFPDoc(const AESTring& docName="", const AESTring& comment="",
           ostream& outputFile=cont, ostream* externalIndexFile=0);
    AFPDoc(const AFPDoc&);
    virtual ~AFPDoc();
    AFPDoc& operator=(const AFPDoc& doc);
    ulong AddPage(const AFPPage& newPage);
    void Comment(const AESTring& comment);
    void BeginGroup(const AESTring& name);
    void EndGroup(void);
    void InvokeMediumMap(const AESTring& mmName);
    void Link(const LLESF& lleStructuredField);
    AESTring Mnemonic(void) const;
    void Tag(const AESTring& attributeName, const AESTring& value);
    void TimeStamp(const time_t time);
    void TimeStamp(const struct timeb* time=0);
    int PreloadObject(AESTring& in_objectName, ObjectType in_objectType,
                     OtherObjectType in_otherObjType );
    int PreloadOverlay(AESTring& in_overlayName);
};
```

### SetDocumentCodepage

```
static void SetDocumentCodepage(const char* codepage, char spaceChar);
```

Specifies the code page name used for encoding character strings, such as names, in the output data stream. For example, the FQN (fully-qualified name) triplet on the BDT (Begin Document) structured field uses this code page for its encoding. This function has no effect on text that is printed in the document by the **PText::AddText** or **PText::RepeatString** calls.

It also specifies a single-byte character that is the code point for a blank in the requested code page. This is normally X'20' for ASCII code pages and X'40' for EBCDIC.

If SetDocumentCodepage is not requested, code page T1V10500 is assumed for all character strings in the output data stream.

### SetInputCodepage

```
static void SetInputCodepage(const char* codepage, char spaceChar);
```

Specifies the code page name that is used for encoding all input character strings, such as names and text, that are parameters on the AFP Toolbox functions. This code page is used for all input character strings unless it is changed with another `SetInputCodepage` request. It also specifies a single-byte character that is the code point for a blank in the requested code page. This is normally 'X'20' for ASCII code pages and 'X'40' for EBCDIC.

If `SetInputCodepage` is not issued, code page T1000850 is assumed for all input character strings.

## SetInputDbCodeset

```
static void SetInputDbCodeset(const char* codeset);
```

Specifies the code set name that is used for encoding input character strings. The code set is used for all input character strings unless it is changed with another `SetInputDbCodepage` request.

## Constructors and Operators

```
AFPDoc(const AString& docName="",
        const AString& comment="",
        ostream& outputFile=cont,
        ostream* externalIndexFile=0);
```

The default constructor takes four parameters as input:

### docName

The name of the document used on the BDT (Begin Document) structured field and can be up to 250 characters. Names longer than 250 characters are truncated.

### comment

A string of up to 250 characters generated in a NOP (No Operation) structured field at the beginning of the output file. Comments longer than 250 characters are truncated.

### outputFile

The output stream where output is placed. The output stream might be a file (**fstream**) or a buffer (**stringstream**), where output is returned to the user program for processing. This stream must have been successfully opened for binary output.

**Note:** In MVS, if you use **fstream**, the output consists of a wrapped AFP structured field. See “Troubleshooting” on page 387 for more information.

### externalIndexFile

is the output file (**fstream**) used to contain indexing information in the form of IEL (Index Element) structured fields. This parameter is currently ignored.

## Public Members of AFPDoc

### AddPage

```
ulong AddPage(const AFPPage& newPage);
```

Places a previously created page object in the AFPDoc object and returns the number of pages in the document up to this point. See “AFPPage” on page 319 for information about creating the page.

### BeginGroup

```
void BeginGroup(const AString& name);
```

Begins a named logical grouping of pages. Page groups can be used for indexing and retrieving information in the document by online viewing products such as AFP Workbench and OnDemand. Page group names can be up to 250 characters. Names longer than 250 characters are truncated. The group name should be unique within a document.

## AFPDoc Class

Pages separated with Begin and End Group can be indexed with the **Attribute Name** and **Attribute Value** parameters of the **Tag** function call.

**Note:** Groups of pages cannot be nested or overlapped (that is, a group must end before another can begin). Attempts to create a nested or overlapped group are ignored.

## Comment

```
void Comment(const AESTring& comment);
```

Specifies a comment built as a NOP (No Operation) structured field at the current point in the document. Comments can be placed before or after page groups or pages and can be intermixed with calls to the Tag and Link functions. The maximum comment length is the maximum output record length minus eight bytes.

**Note:** Each use of the Comment function creates a NOP structured field of its own.

## EndGroup

```
void EndGroup(void);
```

Identifies the end of a group of pages in the output document. Attempts to end a group that has not been opened are ignored.

## InvokeMediumMap

```
void InvokeMediumMap(const AESTring& mmName);
```

Gives the eight character name of the new copy group (medium map) for use in an IMM (Invoke Medium Map) structured field. The IMM is used by the print server (PSF) to switch copy groups in the current form definition. Copy groups control output characteristics such as input bin, duplexing, and so on. The copy group identified by the **mmName** parameter must be defined in the form definition used to print or view the document or an error occurs when it is printed. The InvokeMediumMap function must be called after all calls to Tag and Link.

For more information about form definitions and copy groups see the *IBM Page Printer Formatting Aid: User's Guide*.

## Link

```
void Link(const LLESF& lleStructuredField);
```

Includes a previously built LLE (Link Logical Element) structured field. The LLE can be used by presentation systems such as the AFP Workbench or postprocessor applications such as OnDemand. Attempts to add a Link to a page group after the InvokeMediumMap function has been called are ignored. Use of the Link function requires knowledge of the structured field interface to the AFP Toolbox C++ Object Library.

## Mnemonic

```
AESTring Mnemonic(void) const;
```

Returns a mnemonic label ("AFP Document") to identify this object class.

## PreloadObject

Identifies an object that is to be preloaded to the printer before the start of printing. This is a document level function to aid printer performance when large or complex objects are included in the document.

```
int PreloadObject(AESTring& in_objectName, ObjectType in_objectType,  
                  OtherObjectType in_otherObjType);
```

## PreloadOverlay

Identifies an object that is to be preloaded to the printer before the start of printing. This is a document level function to aid printer performance when large or complex overlays are included in the document.

```
int PreloadOverlay(AEString& in_overlayName);
```

## Tag

```
void Tag(const AEString& attributeName, const AEString& value);
```

Provides an attribute name and value pair for building a TLE (Tag Logical Element) structured field. The TLE can be used by presentation systems such as the AFP Workbench or postprocessor applications such as OnDemand. Attempts to add a Tag to a page group after the InvokeMediumMap function has been called are ignored.

## TimeStamp

```
void TimeStamp(const time_t = 0);  
void TimeStamp(const struct timeb* time);
```

Writes a text formatted time stamp (in a NOP structured field) to the document. If TimeStamp is called with no parameter or time value, the current time is used.

There are two forms of the TimeStamp function:

- The **time** value in the first function is an integer time value representing the number of seconds elapsed since Jan 1, 1970 GMT. This value can be obtained by calling the standard C library function **time()**.
- The **timeb** structure is a more accurate representation of a time value and can be obtained by calling the standard C library function **localtime()**.

The output time string is in the following format:

```
Fri Jan 2  
1:46:40.000 1970
```

## Page Class

The Page class is a representation of a page within a Document object.

This section describes the members of the Page class and the operations that are common to all classes that are derived from Page. Page is an abstract class that serves as the base class to all types of Page objects in AFP Toolbox.

## Declarations for the Page Class

To use the Page class functions, specify the following in your program:

```
#define INCL_PAGE
#include <afptk.h>
```

The above statements cause AFP Toolbox to include the appropriate header files at compile time. The following is an excerpt from the **page.h** file that shows the relevant declarations for the members of Page.

```
class Page
{
public:
    Page(void);
    virtual ~Page();
    int operator==(const Page&) const;
    virtual Page* CopyPage(void) const;
    *boolean IsValid(void) const;
    AESTring Mnemonic(void) const;
};
```

## Constructors and Operators

```
public:
    Page(void);
    int operator==(const Page&) const;
```

The default constructor takes no parameters as input.

Operators are defined for comparing pages to one another.

## Public Members of Page

### CopyPage

```
virtual Page* CopyPage(void) const;
```

Creates a dynamically allocated, properly typed copy of a Page object. This function must be implemented by all sub-classes of the Page class.

### IsValid

```
boolean IsValid(void) const;
```

Checks all of the objects within the AFP page to verify that it is valid MO:DCA data.

### Mnemonic

```
AESTring Mnemonic(void) const;
```

Returns the string "Page" to describe this object.

## AFPPage

The AFPPage class is derived from the Page object and supports the creation of AFP pages within AFP documents. It also contains all of the data stream encoding knowledge and state data necessary to render the page in the MO:DCA data stream. This section describes the use of AFPPage.

## Declarations for the AFPPage Class

To use the AFPPage class functions, specify the following in your program:

```
#define INCL_PAGE
#include <afptk.h>
```

The above statements cause AFP Toolbox to include the appropriate header files at compile time. The following is an excerpt from the **page.h** file that shows the relevant declarations for the members of AFPPage.

```
class AFPPage: public Page
{
    friend ostream& operator<<(ostream&, const AFPPage&);

public:
    AFPPage(const AString& pageName="");
    AFPPage(const AString& pageName, ulong pageWidth, ulong pageDepth);
    AFPPage(const AFPPage& page);
    virtual ~AFPPage();
    int operator==(const AFPPage& mPage);
    AFPPage& operator=(const AFPPage& page);
    void Comment(const AString& comment);
    virtual Page* CopyPage(void) const;
    void IncludeBarCode(const BOCA& newBarCodeObject);
    void IncludeImage(const IOCA& newImageObject);
    void IncludeGraphic(const GOCA& newGraphicObject);
    void IncludeGraphicInline(const AString& graphicFilename
                               ulong xOriginOffset, ulong yOriginOffset);
    void IncludeImageInline(AString& imageFilename, ulong xPos,
                            ulong yPos);
    void IncludeObject(const AString& objName, ObjectType objType,
                       ulong width, ulong depth,
                       ulong xOriginOffset = 0xFFFFFUL,
                       ulong yOriginOffset = 0xFFFFFUL,
                       DegRot orientation = DEG_DEFAULT,
                       MappingOption mapping = SCALE_TO_FIT_MOPT,
                       ulong xContentOriginOffset = 0xFFFFFUL,
                       ulong yContentOriginOffset = 0xFFFFFUL);
    void IncludeOverlay(const AString& name, ulong xPos=0, ulong yPos=0,
                       PageOverlayCP overlayType = NORMAL_POCP);
    void IncludePageSegment(const AString& name, ulong xPos=0, ulong yPos=0,
                            UseFrequency freq = LOW_UF);
    void RotateOverlay(const AString& name,
                       ulong xPos = 0, ulong yPos = 0,
                       DegRot rotation = DEG_0,
                       PageOverlayCP overlayType = NORMAL_POCP);
    void ShadeBlock(uint xpos, uint ypos,
                    uint xsize, uint ysize,
                    ushort percentBlack=0);
    void Tag(const AString& attributeName, const AString& value);
    void IncludeText(const PText& newTextObject, boolean adopt);
    void Link(const LLESF& lleStructuredField);
    void Link(ulong xSourceOffset, ulong ySourceOffset,
              ulong sourceWidth, ulong sourceDepth, boolean isExecutable,
              const AString& targetName, char* targetParms);
    AString Mnemonic(void) const;
    AString PageName(void) const;
    void PageName(const AString& name);
    void PageSize(ulong pageWidth, ulong pageHeight);
```

## AFPPage Class

```
int KeepObject(AEString& in_objectName, ObjectType in_objectType,
               OtherObjectType in_otherObjType);
int SetObjectColorProfile(ColorProfile in_colorProfile, AEString& in_objectName,
                          ObjectType in_objectType, OtherObjectType in_otherObjType);
};
```

## Constructors and Operators

```
AFPPage(const AEString& name="");
AFPPage(const AEString& name,
        ulong pageWidth,
        ulong pageHeight);
AFPPage(const AFPPage& page)

AFPPage& operator=(const AFPPage& page);
```

The default constructor takes as input only the page name, and builds an empty AFPPage object with the following characteristics: unit base of inches, units per unit base of 1440 in both X and Y dimensions, and a page size of 8.5 by 11 inches.

The alternate constructor accepts the page name and page dimensions in 1440ths of an inch. The maximum allowable page size is 22 inches by 22 inches.

The assignment operator is used to copy AFPPage objects.

## Public Members of AFPPage

### Comment

```
void Comment(const AEString& comment);
```

Specifies a comment built as a NOP (No Operation) structured field before the BPG (Begin Page) structured field for this page. More than one comment can be built for a given page; they are placed in the output dataset in the same order as the Comment functions are issued. The maximum string length is the maximum output record length minus eight bytes.

**Note:** Each use of the Comment function creates a NOP structured field of its own.

### CopyPage

```
virtual Page* CopyPage(void) const;
```

Creates a dynamically allocated, properly typed copy of a Page object. This function must be implemented for all sub-classes of the AFPPage class.

### IncludeBarCode

```
void IncludeBarCode(const BOCA& newBarCodeObject);
```

Includes a previously built BCOCA object in the page object. See “Bar Code Object Content Architecture (BCOCA)” on page 342 for information about building bar code objects.

### IncludeBarCodeInline

```
void IncludeBarCodeInline(const BOCA& newBarCodeObject);
```

Reads a previously created BCOCA object and includes it inline in the data stream at the specified position. The bar code file name must be fully qualified and be accessible with appropriate permission for this application to read the file. The bar code can reside in a page segment or it can be a standalone object. See “Bar Code Object Content Architecture (BCOCA)” on page 342 for information about building bar code objects.



**IncludeGraphic**

```
void IncludeGraphic((const GOCA& newGraphicObject);
```

Include a previously built GOCA object in the page object. See “Graphic Object Content Architecture (GOCA)” on page 346 for information about building graphic objects.

**IncludeGraphicInline**

```
void IncludeGraphicInline(const AString& graphicFilename,
                          ulong xPos, ulong yPos);
```

Reads a previously created GOCA object and includes it inline in the data stream at the specified position. The graphic file name must be fully qualified and be accessible with appropriate permission for this application to read the file. The graphic can reside in a page segment or it can be a standalone object. The **xPos** and **yPos** parameters are specified in 1440ths of an inch and must specify a position that lets the entire image to fit on the page. If the coordinates specified are off the page, or cause the image to fall off the page, an error occurs when it is printed. You can include up to 252 images per page. See “Image Object Content Architecture (IOCA)” on page 337 for information about building image objects.

**IncludedImage**

```
void IncludeImage(const IOCA& newImageObject);
```

Includes a previously built IOCA image object in the page object. You can include up to 252 images per page. See “Image Object Content Architecture (IOCA)” on page 337 for information about building image objects.

**IncludedImageInline**

```
void IncludeImageInline(AString& imageFilename,
                        ulong xPos, ulong yPos);
```

Reads a previously created IOCA or IM1 image and includes it inline in the data stream at the specified coordinates. The image file name must be fully qualified and be accessible with appropriate permission for this application to read the file. The image can reside in a page segment, but it must be the only image in the file or else all images in the page segment are included at the same position on the page interfering with each other. The **xPos** and **yPos** parameters are specified in 1440ths of an inch and must specify a position that lets the entire image to fit on the page. If the coordinates specified are off the page, or cause the image to fall off the page, an error occurs when it is printed. You can include up to 252 images per page. See “Image Object Content Architecture (IOCA)” on page 337 for information about building image objects.

**IncludeObject**

```
void IncludeObject(const AString& objName, ObjectType objType,
                  ulong width, ulong depth,
                  ulong xOriginOffset = 0xFFFFFUL,
                  ulong yOriginOffset = 0xFFFFFUL,
                  DegRot orientation = DEG_DEFAULT,
                  MappingOption mapping = POSITION_AND_TRIM_MOPT,
                  ulong xContentOriginOffset = 0xFFFFFUL,
                  ulong yContentOriginOffset = 0xFFFFFUL);
```

Includes a previously created AFP resource or other supported data object on the page by building a reference to the object in the data stream. This function generates an IOB (Include Object) structured field. The object name can be up to 250 characters long. Names longer than 250 characters are truncated.

The **objType** parameter specifies the type of the referenced object. The valid object types are:

**PAGE\_SEGMENT\_OT**

A MO:DCA-P page segment resource object containing an IOCA image. Page segment objects containing IM1 image cannot take advantage of the capabilities of IncludeObject and should be referenced with IncludePageSegment instead.

## AFPPage Class

**Note:** Only IOCA page segments can be trimmed or scaled. Trimming and scaling page segments that contain IM1 image data is not supported.

### GRAPHICS\_OT

A graphics object with MO:DCA syntax as defined in the *Graphics Object Content Architecture Reference*.

### IMAGE\_OT

An image object with MO:DCA syntax as defined in the *Image Object Content Architecture Reference*.

### BAR\_CODE\_OT

A bar code object with MO:DCA syntax as defined in the *Bar Code Object Content Architecture Reference*.

### OTHER\_OBJECT\_DATA\_OT

Object data included that is not defined by an IBM presentation architecture. Most “other” types of objects cannot take advantage of the mapping option functions of IncludeObject.

Refer to *Mixed Object Document Content Architecture Reference* for more information about these data object types.

The **width** and **depth** parameters specify the dimensions of the object area to contain the data object. A value of 0 for both of these parameters indicates the object area dimensions specified in the data object are used.

The **xOriginOffset** and **yOriginOffset** parameters specify the position of the upper left hand corner of the object area. A value for either of these parameters of 0xFFFFF indicates that the object area origin offset specified in the data object is used.

The **orientation** parameter specifies the amount of clockwise rotation of the object area's X axis around the defined origin. Valid values for this parameter are:

DEG\_0  
DEG\_90  
DEG\_180  
DEG\_270  
DEG\_DEFAULT

Specifying the value **DEG\_DEFAULT** for this parameter indicated that the orientation specified in the data object is used.

The **mapping** parameter specifies how the data in the referenced object is placed in the object area. Valid values for the **mapping** parameter are:

### POSITION\_MOPT

Position the upper left corner of the object's presentation space with the object's content origin.

### POSITION\_AND\_TRIM\_MOPT

Position the object at the location specified with ObjectAreaPosition within the dimensions specified with ObjectAreaSize and trim what falls outside the area.

### SCALE\_TO\_FIT\_MOPT

Center the object within the area dimension specified with ObjectAreaSize and scale the object to fit within the area.

### CENTER\_AND\_TRIM\_MOPT

Center the object within the area dimension specified with ObjectAreaSize and trim what falls outside the area.

### MIGRATION\_MAPPING\_41\_MOPT

This mapping is for IM image migration. The origin of the IO image presentation space is positioned at the origin of the object area. Each image point in the presentation space is mapped to a presentation device pel. Any portion of the image that falls outside the object area is trimmed.

**MIGRATION\_MAPPING\_42\_MOPT**

This mapping is for IM image migration. The origin of the IO image presentation space is positioned at the origin of the object area. Each image point in the presentation space is doubled in both directions, resulting in four new image points. The four new image points are then mapped to presentation device pels. Any portion of the image that falls outside the object area is trimmed.

**REPLICATE\_AND\_TRIM\_MOPT**

Positions the data object presentation space in the object area so that its origin is coincident with the origin of the object area and its size is unchanged. The data object presentation space, or graphics presentation space window in the case of GOCA, is then replicated in the X and Y directions of the object area until the object area is filled.

The **xContentOriginOffset** and **yContentOriginOffset** parameters specify the offset within the object area of the data object content. These parameters are only used when the **mapping** parameter is set to **POSITION\_MOPT** or **POSITION\_AND\_TRIM\_MOPT**. Otherwise they are ignored. A value for either of these parameters of 0xFFFFF indicates that the content origin offset specified in the data object is used.

**Note:** The IncludeObject function requires appropriate PSF support.

**IncludeOverlay**

```
void IncludeOverlay(const AString& name, ulong xPos=0 ,
                  ulong yPos=0,
                  PageOverlayType overlayType = NORMAL_POCP);
```

Builds a reference to the overlay in the page object and positions it at the specified coordinates. This generates Include Page Overlay (IPO) and Map Page Overlay (MPO) structured field. The overlay name is eight characters long and must match the name of the overlay member used at print time. The **xPos** and **yPos** parameters are specified in 1440ths of an inch and must specify a position that lets the entire overlay to fit on the page. If the coordinates specified are off the page or cause the overlay to fall off the page, an error occurs when it is printed.

You can include up to 127 unique page overlays on a page.

AFP Toolbox does not verify the existence of the overlay in the library or try to read the overlay contents.

**IncludePageSegment**

```
void IncludePageSegment(const AString& name,
                      ulong xPos=0, ulong yPos=0,
                      UseFrequency frequency = LOW_UF);
```

Builds a reference to the page segment in the page object. This generates an IPS (Include Page Segment) structured field and, if the frequency is set to **HIGH\_UF**, it generates a reference to the page segment within a MPS (Map Page Segment) structured field.

The segment name is eight characters long and must match the name of the page segment member used at print time. The **xPos** and **yPos** parameters are specified in 1440ths of an inch and must specify a position that lets the entire page segment to fit on the page. If the coordinates specified are off the page or cause the page segment to fall off the page, an error occurs when it is printed.

The **frequency** parameter indicates how often this segment is printed in this document. The valid values for this parameter are:

```
LOW_UF  = 0x00,
HIGH_UF = 0xFF
```

AFP Toolbox does not verify the existence of the page segment in the library or try to read the segment contents.

## AFPPage Class

### IncludeText

```
void IncludeText(const PText& newTextObject, boolean adopt);
```

Includes a previously built PText (text) object in the page object. Adopt places a text object directly on the page instead of copying it which improves performance. When adopt is used, the object is owned by the page and is deleted when the page object is deleted. See “PText” on page 326 for information about how to build the text object.

### KeepObject

Identifies an object that is to be kept at the printer while the page is printing. This is a page level function to increase printer performance when large or complex objects are included in a page.

```
int KeepObject(AEString& in_objectName, ObjectType in_objectType,  
              OtherObjectType in_otherObjType);
```

### Link

```
void Link(const LLESF& lleStructuredField);  
void Link(ulong xSourceOffset, ulong ySourceOffset,  
          ulong sourceWidth, ulong sourceDepth, boolean isExecutable,  
          const AEString& targetName, char* targetParms);
```

Creates or includes an LLE (Link Logical Element) structured field. The LLE can be used by presentation systems such as the AFP Workbench or postprocessor applications such as OnDemand.

The first form of the Link function includes a previously built LLE structured field. Use of the Link function requires knowledge of the structured field interface to the AFP Toolbox C++ Object Library. For more information about using the structured field objects, see “The AFP Toolbox C++ Object Library” on page 5.

The second form of the Link function creates an LLE structured field from parameters. The **xSourceOffset** and **ySourceOffset** parameters specify the coordinates of the upper left hand corner of the Link source area in 1440ths of an inch. The **sourceWidth** and **sourceDepth** parameters specify the dimensions of the source area in 1440ths of an inch.

If **isExecutable** is set to **TRUE**, it specifies that the Link target is an executable program.

The **targetName** parameter specifies the target of a link and can identify an executable program or the name of another document. The **targetParms** parameter specifies any additional parameters to pass to the target (if the target is an executable program). For example, to define a link from any point on this page to the document *REFERNC.AFP*, use the following call:

```
pageObj.Link(0,0,IN_2_U1440(8.5),IN_2_U1440(11),FALSE,"REFERNC.AFP","").
```

When viewing this page with the AFP Workbench, this link would cause a separate copy of the AFP Workbench to start to view the document *REFERNC.AFP*

### Mnemonic

```
AEString Mnemonic(void) const;
```

Returns the string “AFPPage” to describe this object.

### PageName

```
AEString PageName(void) const;  
void PageName(const AEString& name);
```

Sets or returns the name for this page. The name is used on the BPG (Begin Page) structured field and can be up to 250 characters. Names longer than 250 characters are truncated.

### PageSize

```
void PageSize(ulong pageWidth, ulong pageHeight);
```

Sets the page size. Dimensions are given in units of 1440ths of an inch. The maximum allowable page size is 22 inches by 22 inches.

### SetObjectColorProfile

Identifies the color profile to be used when a specified object is printed. This is a page level function to ensure the color integrity of an object when it is included in a page.

```
int SetObjectColorProfile(ColorProfile in_colorProfile, AString& in_objectName,
                        ObjectType in_objectType, OtherObjectType in_otherObjType );
```

### Tag

```
void Tag(const AString& attributeName, const AString& value);
```

Provides an attribute name and value pair for building a TLE (Tag Logical Element) structured field. The TLE can be used by presentation systems, such as the AFP Workbench, or postprocessor applications, such as OnDemand.

The **attributeName** parameter identifies the name of the tag. This is often used to specify an attribute of the data being tagged, such as “Customer Name” or “Policy Number”.

The **value** parameter specifies the string that is the particular value of a data attribute. In the previous example, values might be “John Doe” and “123-456-789”.

### RotateOverlay

```
void RotateOverlay(const AString& name,
                  ulong xPos = 0, ulong yPos = 0,
                  DegRot rotation = DEG_0,
                  PageOverlayCP overlayType = NORMAL_POCP);
```

Include and rotate an overlay at the specified position. The **XPos** and **YPos** parameters are the coordinates, in 1440ths of an inch, of the upper left-hand corner of the object area. The **rotation** parameter specifies the clockwise rotations of the overlay area from the X axis rotation of the page. This generates Include Page Overlay (IPO) and Map Page Overlay (MPO) structured fields. The overlay name is eight characters long and must match the name of the overlay member used at print time. If the coordinates specified are off the page or cause the overlay to fall off the page, an error occurs when it is printed.

You can include up to 127 unique page overlays on a page.

AFP Toolbox does not verify the existence of the overlay in the library or try to read the overlay contents.

The **rotation** parameter specifies the amount of clockwise rotation of the overlay. Valid values for this parameter are:

```
DEG_0
DEG_90
DEG_180
DEG_270
```

### ShadeBlock

```
void ShadeBlock(uint xpos, uint ypos,
                uint xsize, uint ysize,
                ushort percentBlack=0);
```

Creates a shaded block at the specified position. The **xpos** and **ypos** parameters are the coordinates, in 1440ths of an inch, of the upper left-hand corner of the shaded area. The **xsize** and **ysize** parameters are the width and depth of the shaded rectangle. Specify the shading value as a percentage from 0 to 100. This function requires microcode support in your printer.

### PText

The Presentation Text (PText) object class encapsulates an AFP text object. The PText class is not derived from the OCA class because it has not been architected as a true OCA object with an OEG and all associated OCA behavior. This Presentation Text object is a device-independent, self-defining representation of a two dimensional presentation space containing the text data.

The unit base used to position all things in the PText object class is 1440ths of an inch.

### Declarations for the PText Class

To use the PText class functions, specify the following in your program:

```
#define INCL_PTEXT
#include <afptk.h>
```

The above statements cause AFP Toolbox to include the appropriate header files at compile time. The following is an excerpt from the ptext.h file that shows the relevant declarations for the members of PText.

```
class PText: public SFContainerUnsorted
{
    friend ostream& operator<<(ostream&, const PText&);

public:
    PText(const AString& PTextName);
    PText(const PText& ptext);
    virtual ~PText();
    int operator==(const PText&) const;
    PText& operator=(const PText& ptext);
    void AddText(const AString& newText);
    void AddText(const AString& newText, short varInc, short adjust);
    void AddText(const AString& newText, short varInc, short adjust,
        AlignmentOption align, char alignmentChar,
        long* rightWidth);
    void AddDbText(const AString& newText);
    void AddDbText(const AString& newText, short varInc, short adjust);
    void AddDbText(const AString& newText, short varInc, short adjust,
        AlignmentOption align, char* alignmentCharPtr,
        long* rightWidth);
    void BeginSuppression(ushort id);
    void Color(MODCAColors color = DEFAULT_COLOR);
    void Controls(const PT1Con& newControl, SeqHandle handle,
        DataEffect effect, Chain chaining);
    void DrawBox(short ruleWidth, short xPosition, short yPosition,
        short width, short depth);
    void DrawRule(short height=0, short width=0, int type, int pattern=0);
    void EndSuppression();
    ostream& Export(ostream& os) const;
    PTDSF ExtractPTD(void);
    const ulong* ExtractFonts(void) const;
    void HMove(ushort xRelative);
    void HMoveTo(short xAbsolutePos);
    void MaxPTXLength(ushort maxLen);
    ushort MaxPTXLength(void) const;
    AString Mnemonic(void) const;
    void Move(short xRelative, short yRelative);
    void MoveTo(ushort xAbsolutePos, ushort yAbsolutePos);
    short LineSpace(void) const;
    void NextLine(void);
    void NextLine(short horizStartPoint);
    void RepeatString(const AString& repeatedString, ushort length);
    void SetFont(ulong nickname);
    void SetTextOrientation(TextOrientation orient=IOB90_T0);
    void VMove(short yRelative);
    void VMoveTo(ushort yAbsolutePos);
};
```

## Constructors and Operators

```
PText(const AString& PTextName);
int operator==(const PText&) const;
PText& operator=(const PText& ptext);
```

The default constructor takes only one parameter, which is the name of the PText object, used on the BPT (Begin Presentation Text) structured field. The name can be up to 250 characters long. Names longer than 250 characters are truncated.

The operators are used for comparing and assigning PText objects.

## Public Members of PText

### AddText

```
void AddText(const AString& newText);
void AddText(const AString& newText, short varInc, short adjust);
void AddText(const AString& newText, short varInc, short adjust,
             AlignmentOption align, char alignmentChar,
             long* rightWidth);
```

Adds a text string to the PText object. There are three forms of AddText.

The first form takes only a text string and uses the font information to determine the spacing between words and characters in the string. The string is left-aligned at the current position.

The second form lets you override the width of a blank (the **varInc** parameter) and add additional intercharacter spacing (the **adjust** parameter). You can specify -1 for the value of either of these parameters to tell AFP Toolbox to use the previously set value or the default value from the font. Both parameters have a valid range of 0 to 32,767 if the default is not used.

The third form also lets you specify how you want the string of text aligned at the current position. The choices are:

LEFT_ALIGN	left aligned
RIGHT_ALIGN	right aligned
CENTER_ALIGN	centered
CHAR_ALIGN	aligned on the specified character
FIRST_CHAR_ALIGN = (CHAR_ALIGN)	align on first occurrence of char
LAST_CHAR_ALIGN,	align on last occurrence of char

In all three cases the text string is formatted in the current font (last specified with SetFont).

The **rightWidth** parameter is an output parameter that, if not null, receives a measurement of the portion of the string to the right of the current position.

**Note:** Changes you make to the variable increment and intercharacter adjustment spacing stay in effect throughout the document. They do not automatically revert to the default.

Because measuring the width of a string is a relatively performance intensive operation and is often not required, it is not routinely done on the AddText operation. The current position (as returned by the QueryPosition call) might not reflect the true current position after adding text. When the **rightWidth** parameter is specified with a non-null value, the text is measured and the current position is updated. If your application requires absolute maintenance of the current position at all times (most applications do not), then always use the third form of this function with a non-null value for the **rightWidth** parameter. For alignments other than **LEFT\_ALIGN**, the string is always measured.



## PText Class

### AddDbText

```
void AddDbText(const AString& newText);  
void AddDbText(const AString& newText, short varInc, short adjust);  
void AddDbText(const AString& newText, short varInc, short adjust,  
               AlignmentOption align, char* alignmentCharPtr,  
               long* rightWidth);
```

Adds an MBCS text string to the PText object. Similar to AddText, there are three forms of AddDbText.

The first form takes only a text string and uses the font information to determine the spacing between words and characters in the string. The string is left-aligned at the current position.

The second form lets you override the width of a blank (the **varInc** parameter) and add additional intercharacter spacing (the **adjust** parameter). You can specify -1 for the value of either of these parameters to tell AFP Toolbox to use the previously set value or the default value from the font. Both parameters have a valid range of 0 to 32,767 if the default is not used.

The third form lets you specify the pointer to an alignment character of a MBCS character. The choices are:

LEFT_ALIGN	left aligned
RIGHT_ALIGN	right aligned
CENTER_ALIGN	centered
CHAR_ALIGN	aligned on the specified character
FIRST_CHAR_ALIGN = (CHAR_ALIGN)	align on first occurrence of char
LAST_CHAR_ALIGN,	align on last occurrence of char

In all three cases the text string is formatted in the current font (last specified with SetFont).

The **rightWidth** parameter is an output parameter that, if not null, receives a measurement of the portion of the string to the right of the current position.

**Note:** Changes you make to the variable increment and intercharacter adjustment spacing stay in effect throughout the document. They do not automatically revert to the default.

Because measuring the width of a string is a relatively performance intensive operation and is often not required, it is not routinely done on the AddText operation. The current position (as returned by the QueryPosition call) might not reflect the true current position after adding text. When the **rightWidth** parameter is specified with a non-null value, the text is measured and the current position is updated. If your application requires absolute maintenance of the current position at all times (most applications do not), then always use the third form of this function with a non-null value for the **rightWidth** parameter. For alignments other than **LEFT\_ALIGN**, the string is always measured.

### Color

```
void Color(MODCColors color = DEFAULT_COLOR);
```

Sets the color for subsequent text and rules in the PText object. Valid values are:

```
BLUE  
RED  
PINK  
MAGENTA  
GREEN  
TURQUOISE  
CYAN  
YELLOW  
BLACK  
BROWN  
MEDIUM_COLOR  
DEFAULT_COLOR
```



**Note:** The color you are able to view is dependent on your display device.

## DrawBox

```
void DrawBox(short ruleWidth, short xPosition, short yPosition,
             short width, short depth);
```

Draws a fixed size box at the specified location for a given width and depth, using **ruleWidth** for the thickness of the rules on all four sides of the box.

The **ruleWidth** parameter is the width, in 1440ths of an inch, of the lines used to draw the box. This parameter might be negative, in which case the rules for the box are drawn outside to the left and above the coordinate points rather than to the right and below. Negative rule widths are not recommended as not all printers can print the resulting boxes.

The **xPosition** and the **yPosition** parameters specify the position, in 1440ths of an inch, of the upper left hand corner of the box. This position must be on the page and at a point so that the entire box fits on the page or an error occurs when it is printed.

The **width** and **depth** parameters specify the dimensions, in 1440ths of an inch, of the box. The width is measured from the right side of the right most rule to the right side of the left rule. The depth is measured from the top of the top rule to the top of the bottom rule. The dimensions for the box must be specified so that at the specified position the entire box fits on the page or else an error occurs when it is printed.

## DrawRule

```
void DrawRule(long height, long width = 0);
              int type = 0, int pattern = 0);
```

Draws a fixed-length horizontal or vertical rule at the current position. If the height is larger than the width, this function draws a vertical rule, otherwise it draws a horizontal rule.

The **height** and **width** parameters are specified in 1440ths of an inch and might be negative. Positive rule dimensions extend to the right and down from the current position. Negative rule dimensions extend to the left and up from the current position. A value of 0 for either the width or height instructs the printer to use the thinnest visible rule (usually 1 pel). Specifying 0 for both parameters causes the function call to be ignored.

The height and width of the rule must be such that from the current position, the entire rule fits on the page or else an error occurs when it is printed.

The pattern and type parameters are currently ignored.

## Hmove

```
void HMove(short xRelative);
```

Moves horizontally, relative to the current position, a specified distance in 1440ths of an inch. Positive values move the current position to the right; negative values move the current position to the left. Relative moves of zero are ignored.

## HMoveTo

```
void HMoveTo(ushort xAbsolutePos);
```

Moves to a specific horizontal location in 1440ths of an inch relative to the page origin.

## MaxPTXLength

```
void MaxPTXLength(ushort maxLen);
ushort MaxPTXLength(void) const;
```

## PText Class

Sets or returns the maximum length of each PTX<sup>®</sup> structured field within the current PText object. Valid range is 120 to 32767 bytes. The default value is 8192.

## Mnemonic

```
AEStrng Mnemonic(void) const;
```

Returns the string "PText" to describe this object.

## Move

```
void Move(short xRelative, short yRelative);
```

Moves both horizontally and vertically relative to the current position to a specified distance in 1440ths of an inch.

The **xRelative** parameter specifies the distance to move horizontally. Positive values move to the right; negative values move to the left. A value of 0 is ignored.

The **yRelative** parameter specifies the distance to move vertically. Positive values move down the page; negative values move up the page. A value of 0 is ignored.

## MoveTo

```
void MoveTo(ushort xAbsolutePos, ushort yAbsolutePos);
```

Moves both horizontally and vertically to a specific location in 1440ths of an inch relative to the page origin.

## RepeatString

```
void RepeatString(const AEStrng& repeatedString, ushort length);
```

Writes a repeated string of characters at the current location in the current font for the number of characters specified by length. The string is repeated as many times as necessary to equal the length of the number of characters. If the length is not an integral multiple of the string of characters, the last repetition of the string is truncated. If the length is less than the number of characters in the string, then the string is truncated.

**Note:** RepeatString writes a string of SBCS characters only. Do not apply RepeatString on a DBCS font.

## SetFont

```
void SetFont(ulong nickname);
```

Selects the font used for subsequent text strings. The font nickname must have been previously defined with the AFPDefineFontByAttributes, AFPDefineFontByAttrWithScaling, or AFPDefineFontByName functions. See Chapter 6, "Defining Fonts with C++ Language Programs" on page 355 for more information about these functions.

## SetTextOrientation

```
void SetTextOrientation(TextOrientation orient=IOB90_TO);
```

Selects the orientation of subsequent text strings in the PText object. Valid values for orientation are as follows:

<b>IOB90_TO</b>	Text is rotated zero degrees clockwise. The text origin is at the upper, left-hand corner of the page.
<b>IOB270_TO</b>	Text is rotated zero degrees clockwise. The text origin is at the lower, left-hand corner of the page.

<b>I90B180_TO</b>	Text is rotated 90 degrees clockwise. The text origin is at the upper, right-hand corner of the page.
<b>I90B0_TO</b>	Text is rotated 90 degrees clockwise. The text origin is at the upper, left-hand corner of the page.
<b>I180B270_TO</b>	Text is rotated 180 degrees clockwise. The text origin is at the lower, right-hand corner of the page.
<b>I180B90_TO</b>	Text is rotated 180 degrees clockwise. The text origin is at the upper, right-hand corner of the page.
<b>I270B0_TO</b>	Text is rotated 270 degrees clockwise. The text origin is at the lower, left-hand corner of the page.
<b>I270B180_TO</b>	Text is rotated 270 degrees clockwise. The text origin is at the lower, right-hand corner of the page.

For example, to rotate the text orientation 90 degrees clockwise, specify I90B180\_TO.

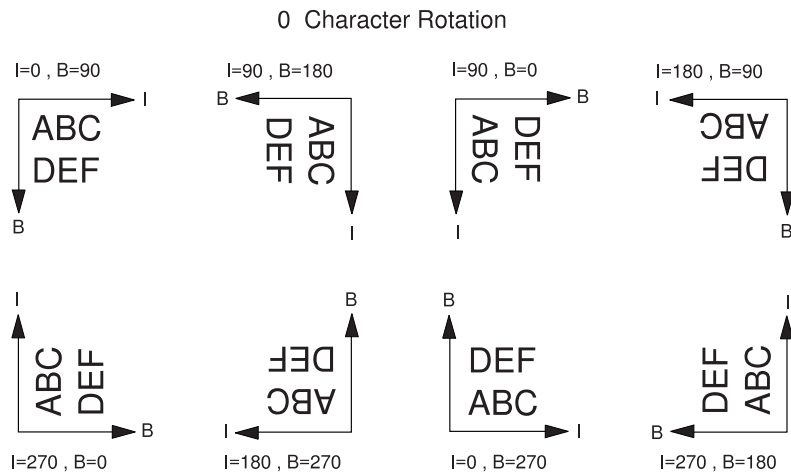


Figure 24. Text Orientation Examples

This function call moves the text origin and coordinate system for all subsequent PText operations. However, it does not effect the positioning or rotation of any image, overlay, bar code, or other data objects or resources.

For example, if the text orientation is set to I90B180\_TO, the origin is moved 90 degrees clockwise about the page (to the top right hand corner). All subsequent coordinate positions are rotated as well. In effect it is as though the page were rotated 90 degrees counterclockwise and all of the measurements still refer to the top left hand corner.

This operation keeps the text and positioning consistent with the perceived orientation of the text on the page.

The symbolic values for the **orient** parameter are constructed from the inline and baseline rotations as shown in Figure 24. For example, to place characters as pictured in the lower right hand corner of the figure (I=270 degrees, B=180 degrees), the value for the **orient** parameter is I270B180\_TO. See Figure 24 for examples of different text orientations.

## Vmove

```
void VMove(short yRelative);
```

## **PText Class**

Moves vertically relative to the current position a specified distance in 1440ths of an inch. Positive values move the current position down the page; negative values move the current position up the page. Relative moves of zero are ignored.

## **VMoveTo**

```
void VMoveTo(ushort yAbsolutePos);
```

Moves to a specific vertical location in 1440ths of an inch relative to the page origin.

## OCA

The Object Content Architecture (OCA) class is an abstract class that is the parent of the IOCA and BCOCA data object class. It contains function to cover the common aspects of MO:DCA data objects, such as the Object Area, Object Environment Group (OEG), and miscellaneous common data items.

The OCA class maintains the data needed to construct the Object Environment Group for its concrete subclasses. This data includes the Object Area Descriptor (OBD) and Object Area Position (OBP) structured fields, and the Map structured fields for the various data object types.

Each concrete data object is mapped into an Object Area as defined by the OBD structured field. The Object Area has a size and position. The size is measured in the terms specified on the area descriptor and the position is measured by the units defined for the page. The data object content can be positioned at a specified point within the Object Area.

The position of the Object Area should be defined by constructor parameters. The other data fields are defaulted based on the assumption of the MO:DCA architecture defaults as applied to a portrait page and nonrotated data object. All data elements can be overridden by member functions.

## Declarations for the OCA Class

To use the OCA class functions, specify the following in your program:

```
#define INCL_OCA
#include "afptk.h"
```

The above statements cause AFP Toolbox to include the appropriate header files at compile time. The following is an excerpt from the oca.h file that shows the relevant declarations for the members of OCA.

```
class OCA
{
    friend ostream& operator<<(ostream&, const OCA&);

public:
    virtual ~OCA();
    virtual int operator==(const OCA&) const;
    virtual OCA* CopyOCA(void) const;
    virtual ostream& Export(ostream&) const;
    virtual OCATypes OCAType(void) const;
    boolean IsValid(void) const;
    AString Mnemonic(void) const;
    void ObjectAreaMeasurementUnits(MUUnitBase ubase = TEN_INCHES_UB,
                                    ushort xUnitsPerBase = 14400,
                                    ushort yUnitsPerBase = 14400,
                                    boolean forPageSegment = FALSE);
    void ObjectAreaMeasurementUnits(MUUnitBase& ubase,
                                    ushort& xUnitsPerBase,
                                    ushort& yUnitsPerBase,
                                    boolean& forPageSegment) const;
    void ObjectAreaMixingFlag(flag mixingFlag);
    flag ObjectAreaMixingFlag(void) const;
    void ObjectAreaPosition(ulong areaXOrigin,
                            ulong areaYOrigin,
                            DegRot areaXOrientation = DEG_0);
    void ObjectAreaPosition(ulong& areaXOrigin,
                            ulong& areaYOrigin,
                            DegRot& areaXOrientation) const;
    void ObjectAreaContentPosition(ulong contentXOrigin,
                                    ulong contentYOrigin);
    void ObjectAreaContentPosition(ulong& contentXOrigin,
                                    ulong& contentYOrigin) const;
    void ObjectAreaSize(ulong xSize, ulong ySize);
```

## OCA Class

```
void ObjectAreaSize(ulong& xSize, ulong& ySize) const;
void ObjectContentMapping(MappingOption mopt);
MappingOption ObjectContentMapping(void) const;
};
```

## Constructors and Operators

This class is currently an abstract class due to its protected default constructor.

## Public Members of OCA

### CopyOCA

```
virtual OCA* CopyOCA(void) const;
```

Returns a properly typed copy of the OCA object. All subclasses of OCA must redefine the function.

### Mnemonic

```
AEStrng Mnemonic(void) const;
```

Returns the string "OCA" to represent this object.

### ObjectAreaContentPosition

```
void ObjectAreaContentPosition(ulong contentXOrigin,
                               ulong contentYOrigin);
void ObjectAreaContentPosition(ulong& contentXOrigin,
                               ulong& contentYOrigin) const;
```

Set or query the area content position for this object. The **contentXOrigin** and **contentYOrigin** parameters are the coordinates, in 1440ths of an inch, offset within the object area of the object content. These parameters are used only when an ObjectContentMapping of **POSITION\_MOPT** or **POSITION\_AND\_TRIM\_MOPT** is used, otherwise they are ignored. See Figure 25 on page 336 for an example of the ObjectAreaContentPosition.

### ObjectAreaMeasurementUnits

```
void ObjectAreaMeasurementUnits(MUUnitBase ubase = TEN_INCHES_UB,
                                ushort xUnitsPerBase = 14400,
                                ushort yUnitsPerBase = 14400,
                                boolean forPageSegment = FALSE);
void ObjectAreaMeasurementUnits(MUUnitBase& ubase,
                                ushort& xUnitsPerBase,
                                ushort& yUnitsPerBase,
                                boolean& forPageSegment) const;
```

Set or query the measurement units for this object. Currently the default values are the only values supported.

### ObjectAreaMixingFlag

```
void ObjectAreaMixingFlag(flag mixingFlag);
flag ObjectAreaMixingFlag(void) const;
```

Set or query the area mixing flag for this object.

**SET=1** Results in a new presentation space covering the existing presentation space. Specifically, the background of the new presentation space overpaints the background and foreground of the existing presentation space, and the foreground of the new presentation space overpaints the background and foreground of the existing presentation space.

**RESET=0** Results in the new presentation space mixing with the existing presentation space in accordance with the default MO:DCA mixing rule. Specifically, the background of the new presentation space underpaints both the background and foreground of the existing

presentation space, and the foreground of the new presentation space overpaints the background and the foreground of the existing presentation space.

### ObjectAreaPosition

```
void ObjectAreaPosition(ulong areaXOrigin,
                        ulong areaYOrigin,
                        DegRot areaXOrientation = DEG_0);
void ObjectAreaPosition(ulong& areaXOrigin,
                        ulong& areaYOrigin,
                        DegRot& areaXOrientation) const;
```

Set or query the area position information for this object. The **areaXOrigin** and **areaYOrigin** parameters are the coordinates, in 1440ths of an inch, of the upper left-hand corner of the object area. The **areaXOrientation** parameter specifies the clockwise rotations of the object area from the X axis rotation of the page. See Figure 25 on page 336 for an example of a the ObjectAreaPosition.

### ObjectAreaSize

```
void ObjectAreaSize(ulong xSize, ulong ySize);
void ObjectAreaSize(ulong& xSize, ulong& ySize) const;
```

Set or query the area size for this object.

### ObjectContentMapping

```
void ObjectContentMapping(MappingOption mopt);
MappingOption ObjectContentMapping(void) const;
```

Set or query the content mapping for this object. The valid values for **mopt** are:

#### POSITION\_MOPT

Position the upper left corner of the object's presentation space with the object's content origin.

#### POSITION\_AND\_TRIM\_MOPT

Position the object at the location specified with ObjectAreaPosition within the dimensions specified with ObjectAreaSize and trim what falls outside the area.

#### SCALE\_TO\_FIT\_MOPT

Center the object within the area dimension specified with ObjectAreaSize and scale the object to fit within the area.

#### CENTER\_AND\_TRIM\_MOPT

Center the object within the area dimension specified with ObjectAreaSize and trim what falls outside the area.

#### MIGRATION\_MAPPING\_41\_MOPT

This mapping is for IM image migration. The origin of the IO image presentation space is positioned at the origin of the object area. Each image point in the presentation space is mapped to a presentation device pel. Any portion of the image that falls outside the object area is trimmed.

#### MIGRATION\_MAPPING\_42\_MOPT

This mapping is for IM image migration. The origin of the IO image presentation space is positioned at the origin of the object area. Each image point in the presentation space is doubled in both directions, resulting in four new image points. The four new image points are then mapped to presentation device pels. Any portion of the image that falls outside the object area is trimmed.

#### REPLICATE\_AND\_TRIM\_MOPT

Position the data object presentation space in the object area so that its origin is coincident with the origin of the object area and its size is unchanged. The data object presentation space, or graphics presentation space window in the case of GOCA, is then replicated in the X and Y directions of the object area until the object area is filled.

## OCA Class

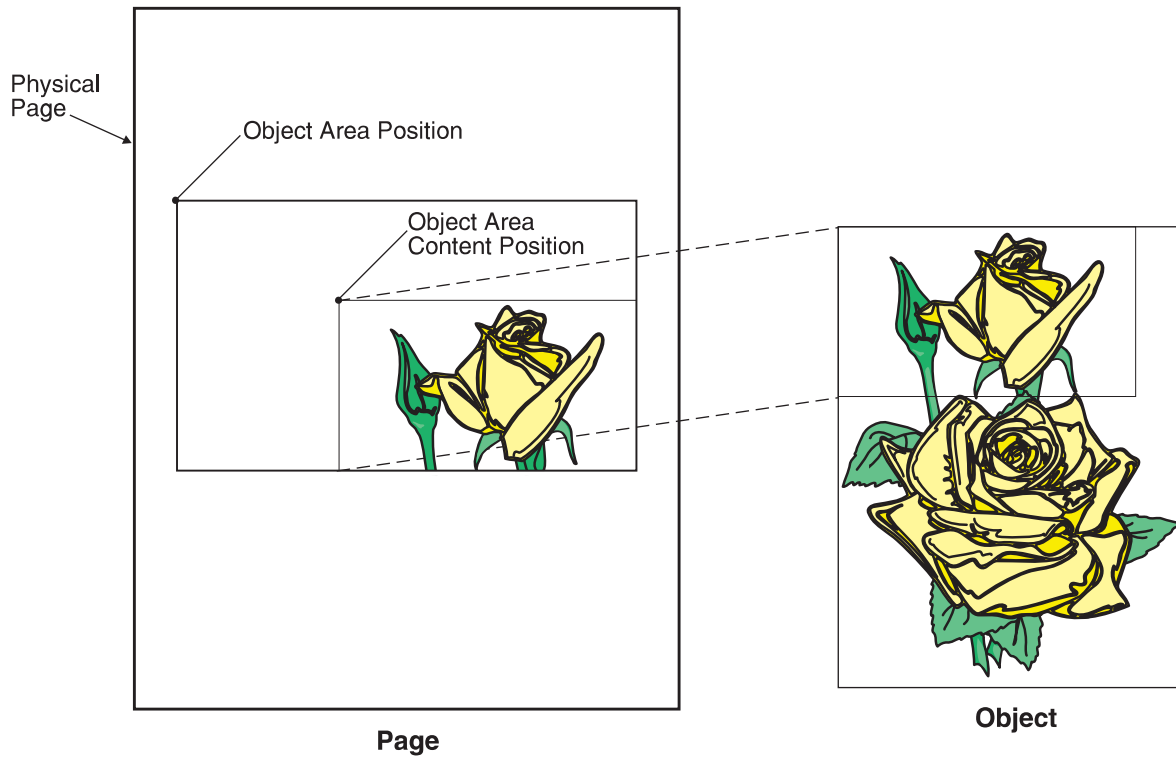


Figure 25. Object and Object Content Positioning. Position and trim (POSITION\_AND\_TRIM\_MOPT) was defined for positioning this object.

Position the data object presentation space in the object area so that its origin is coincident with the origin of the object area and its size is unchanged. The data object presentation space or graphics presentation window in the case of GOCA is then replicated in the X and Y directions of the object area until the object area is filled.

## OCAType

```
virtual OCATypes OCAType(void) const;
```

Returns the type of the OCA object, for example IOCA or GOCA.



## Image Object Content Architecture (IOCA)

The Image Object Content Architecture (IOCA) object class encapsulates an AFP image object. This Image object is a device-independent, self-defining representation of a two-dimensional presentation space containing the image data.

Currently, only simple image data is supported. Banded, numbered, and tiled images are not supported.

Because of the containerized nature of IOCA image data, several mandatory self defining fields are created when the `BeginImage` function is called. This also means that when the object is output, the mandatory self defining fields that end the image content and image segment must be appended with the `EndImage` function. Therefore, once ended, no more data can be added to the object, and no parameters can be changed. In addition, once data has been added, no image parameters can be added or changed. See the *Image Object Content Architecture Reference* for more information about IOCA image objects.

The typical sequence of image object function calls is where `[]` indicates optional calls and `...` indicates a call can be made more than once.

```
[ MaxIPDLength ]
  BeginImage
[ IDSize ]
[ IDStructure ]
[ ImageEncoding ]
[ ImagePresentationSpace ]
[ LookUpTable ]
  ImageData ...
  EndImage
```

## Declarations for the IOCA Class

To use the IOCA class functions, specify the following in your program:

```
#define INCL_IOCA
#include <afptk.h>
```

The above statements cause AFP Toolbox to include the appropriate header files at compile time. The following is an excerpt from the **ioca.h** file that shows the relevant declarations for the members of IOCA.

```
class IOCA: public OCA, public SFContainerUnsorted
{
  friend ostream& operator<<(ostream&, const IOCA&);

public:
  IOCA(const AString& IOCAName);
  IOCA(const AString& IOCAName, ushort ubase, uint hr, uint vr,
        uint hs, uint vs, CompressionType cmpr, byte* idat, ulong len);
  IOCA(const IOCA& ioca);
  virtual ~IOCA();
  IOCA& operator=(const IOCA& ioca);
  int operator==(const IOCA&) const;
  void BeginImage(ushort ubase, uint hr, uint vr, uint hs, uint vs);
  virtual OCA* CopyOCA(void) const;
  void EndImage(void);
  ostream& Export(ostream& os) const;
  void IDSize(ushort numIDEBits);
  void IDStructure(flag asf=RESET, flag gc=RESET,
                  COLORModel color=YCrCb, ushort c1bits=0x01,
                  ushort c2bits=0x00, ushort c3bits=0x00);
  void ImageData(byte* imageData, uint imageLength);
  void ImageEncoding(CompressionType cmpr, ushort recordingAlgorithm,
                    ushort bitOrder);
  void ImagePresentationSpace(ushort ubase, uint hr, uint vr,
                              uint hs, uint vs);
  void LookUpTable(ushort lookUpTableID);
  void MaxIPDLength(ushort maxLen);
```

## IOCA Class

```
    ushort MaxIPDLength(void) const;
    AESTring Mnemonic(void) const;
    OCATypes OCAType(void) const;
    void SDFs(const ImageSDF& newSDF, SeqHandle handle,
              DataEffect effect);
};
```

## Constructors and Operators

```
IOCA(const AESTring& IOCAName);
IOCA(const AESTring& IOCAName, ushort ubase, uint hr, uint vr,
      uint hs, uint vs, CompressionType cmpr, byte* idat, ulong len);
IOCA(const IOCA& ioca);
virtual ~IOCA();
IOCA& operator=(const IOCA& ioca);
int operator==(const IOCA&) const;
```

The default constructor takes only one parameter, which is the name of the IOCA object, used on the BIM (Begin Image Object) structured field. The name can be up to 250 characters long.

The long form constructor can be used to create an entire image with one declaration. The **name** parameter is as described above. The **ubase** parameter defines the base units for the other measurement parameters. The valid values for the **ubase** parameter are:

```
TEN_INCHES_UB
TEN_CENTIMETERS_UB
```

The **hr** and **vr** parameters are the horizontal and vertical resolutions of the image respectively. For example, for an image with a resolution of 240 pels per inch in both directions, specify 2400 for **hr** and **vr** (2400 pels in ten inches).

The **hs** and **vs** parameters are the size, in pel, of the image data. For an image two inches square at a resolution of 300 pels per inch, the **hs** and **vs** values would be 600.

The **cmpr** parameter specifies the compression algorithm used to encode the image data. The valid values for this parameter are:

```
MMR      /* IBM MMR (modified CCITT READ Algorithm) */ RAW      /* No compression */
RL4      /* Run-length 4 */
ABIC     /* Bi-level Q-coder */
CABIC    /* Concatenated ABIC */
CCOMP    /* Color compression used by OS/2 */
G3MH     /* CCITT T.4 G3 (1D fax coding) */
G3MR     /* CCITT T.4 G3 (2D fax coding) */
G4MMR    /* CCITT T.6 G4 (2D fax coding) */
JPEG     /* ISO/CCITT JPEG algorithms */
```

This function will not perform the specified compression on the image data. This parameter specifies what compression has already been used to encode the image data.

The **idat** and **len** parameters are the location and length in bytes, respectively, of the image data. The data buffer must contain the complete image and must be of the appropriate length. Due to the differences of the various compression algorithms the **length** parameter is not verified.

Upon construction with this form of the constructor, the image object is complete and no other operation besides the output operator or Export function can be called on the resulting object or else the resultant image is not valid.

The copy constructor duplicates this image object and the operators are used for comparing and assigning image objects.

## Public Members of IOCA

### BeginImage

```
void BeginImage(ushort ubase, uint hr, uint vr, uint hs, uint vs);
```

The **BeginImage** function generates the **BeginSegment**, **BeginImageContent**, and **ImageSize** parameters and adds them to the IOCA object. Once **BeginImage** is called, you can specify the encoding, IDE size, IDE structure, and so forth with the other member functions or add data to the image with the **ImageData** function.

### CopyOca

```
virtual OCA* CopyOca(void) const;
```

Returns a properly typed copy of this object. All subclasses of IOCA must reimplement this function.

### EndImage

```
void EndImage(void);
```

The **EndImage** function generates the **EndImageContent** and **EndSegment** parameters and adds them to the IOCA object. After this function is called no parameters can be added to the image.

### Export

```
ostream& Export(ostream& os) const;
```

Outputs the IOCA object to the specified output stream. The stream specified must have been previously opened for binary output.

### IDESize

```
void IDESize(ushort numIDEbits);
```

Sets the number of bits per Image Data Element.

### IDEStructure

```
void IDEStructure(flag asf = RESET, flag gc = RESET,
                  COLORModel color = YCrCb, ushort c1bits = 0x01,
                  ushort c2bits = 0x00, ushort c3bits = 0x00);
```

Defines the structure of an Image Data Element.

The **asf** parameter indicates whether successive IDE values correspond to brighter or darker levels of gray or color. Specifying **RESET**(0) indicates that successive IDE values represent brighter levels of gray or color. Specifying **SET** (1) indicates darker grays or colors.

The **gc** parameter specifies whether the data uses the Gray coding scheme (**SET**) to encode the image data or does not use it (**RESET**).

The **color** parameter specifies the color model used for the image data. The valid values are:

**RGB** Each value is treated as a set of red, green, and blue intensity values, in that order.

#### YCrCb

Each value is treated as a set of intensity (Y) and chrominance differences (Cr and Cb), in that order.

#### YCbCr

Each value is treated as a set of intensity (Y) and chrominance differences (Cb and Cr), in that order.

## IOCA Class

The **c1bits**, **c2bits**, and **c3bits** parameters specify the number of bits in each of the 3 color components. For example, for 24 bit RGB color, set each of these values to 8.

## ImageData

```
void ImageData(byte* imageData, uint imageLength);
```

Adds image data to the IOCA object. The ImageData and ImageLength functions specify the location and length in bytes of an encoded image data buffer. Once the ImageData function has been called only another ImageData or EndImage function can be called. No other image parameters can be set.

## ImageEncoding

```
void ImageEncoding(CompressionType cmpr,
    ushort recordingAlgorithm,
    ushort bitOrder);
```

Sets image encoding for the IOCA object. The **cmpr** parameter specifies the compression algorithm used to encode the image data. The valid values for this parameter are:

```
MMR      /* IBM MMR
(modified CCITT READ Algorithm */ RAW      /* No compression */
RL4      /* Run-length 4 */
ABIC     /* Bi-level Q-coder */
CABIC    /* Concatenated ABIC */
CCOMP    /* Color compression used by OS/2 */
G3MH     /* CCITT T.4 G3 (1D fax coding) */
G3MR     /* CCITT T.4 G3 (2D fax coding) */
G4MMR    /* CCITT T.6 G4 (2D fax coding) */
JPEG     /* ISO/CCITT JPEG algorithms */
```

This function does not perform the specified compression on the image data. This parameter specifies what compression has already been used to encode the image data.

The **recordingAlgorithm** parameter specifies how the image data is recorded. The valid values are:

- 1** RIDIC (Recording Image Data Inline Coding)
- 3** Bottom to top

The **bitOrder** parameter specifies the bit order within each image data byte. A value of 0 indicates left to right order. A value of 1 indicates right to left order. All other values are invalid and result in a printing error.

The **bitOrder** parameter is ignored because it only applies to IOCA Function Set 11 (FS11). The Toolbox currently supports only Function Set 10 (FS10) because not all IPDS printers support FS11.

## LookupTable

```
void LookupTable(ushort lookupTableID);
```

Specifies the look up table identifier to use to interpret the image data. Each IDE value is an index in this LUT. Valid values for this parameter are from 0-255.

## MaxIPDLength

```
void MaxIPDLength(ushort maxLen);
ushort MaxIPDLength(void) const;
```

Sets or queries the maximum length of each IPD structured field within the current IOCA object. The valid range for this parameter is 120-32759 bytes. The default value is 32759.

## Mnemonic

```
AEString Mnemonic(void) const;
```

Returns the string "IOCA" to describe this object.

### Bar Code Object Content Architecture (BCOCA)

The Bar Code Object Content Architecture (BCOCA) object class encapsulates an AFP bar code object. This object is a device-independent, self-defining representation of a two-dimensional presentation space containing the bar code data (see Figure 7 on page 33).

### Declarations for the BCOCA Class

To use the BCOCA class functions, specify the following in your program:

```
#define INCL_BCOCA
#include <afptk.h>
```

The above statements cause AFP Toolbox to include the appropriate header files at compile time. The following is an excerpt from the **bcoca.h** file that shows the relevant declarations for the members of BCOCA.

```
class BCOCA: public OCA, public SFContainerUnsorted
{
friend ostream& operator<<(ostream&, const BCOCA&);

public:
    BCOCA(const AString& BCOCAName);    // 3-of-9 code, no check, 8.5x11
    BCOCA(const AString& BCOCAName,    // 8.5 x 11 space in 1440ths
           BarCodeType type, ushort mod, // bar code type and modifier
           FontID font = 0,             // HRI font (0 = default)
           MODCAColors color = NEUTRAL_WHITE, // bar code color
           ushort moduleWidth = 0xFF,    // FF is the device default
           ushort elementHeight = 0xFFFF,
           ushort heightMultiplier = 1,
           ushort wideNarrowRatio = 0xFFFF);
    BCOCA(const AString& BCOCAName, // bar code name
           MUUnitBase ubase,        // unit base
           ushort xupb, ushort yupb, // units per base
           ushort xSize, ushort ySize, // dimensions of presentation space
           BarCodeType type, ushort mod, // bar code type and modifier
           FontID font = 0,
           MODCAColors color = NEUTRAL_WHITE,
           ushort moduleWidth = 0xFF,
           ushort elementHeight = 0xFFFF,
           ushort heightMultiplier = 1,
           ushort wideNarrowRatio = 0xFFFF);
    BCOCA(const BCOCA& bcoca);
    virtual ~BCOCA();
    BCOCA& operator=(const BCOCA& bcoca);
    int operator==(const BCOCA&) const;
    void AddBarCode(ushort xPos, ushort yPos, const AString& data,
                   boolean hri = TRUE, boolean hriOnTop = FALSE,
                   boolean hriOnBottom = FALSE, boolean asterisk = FALSE);
    virtual OCA* CopyOCA(void) const;
    ostream& Export(ostream& os) const;
    FontID Font(void) const;
    AString Mnemonic(void) const;
    OCATypes OCAType(void) const;
};
```

### Constructors and Operators

```
BCOCA(const AString& BCOCAName);
BCOCA(const AString& BCOCAName,
       BarCodeType type, ushort mod,
       FontID font = 0,
       MODCAColors color = NEUTRAL_WHITE,
       ushort moduleWidth = 0xFF,
       ushort elementHeight = 0xFFFF,
       ushort heightMultiplier = 1,
```

```

        ushort wideNarrowRatio = 0xFFFF);
BCOCA(const AString& BCOCAName,
        MUUnitBase ubase,
        ushort xupb, ushort yupb,
        ushort xSize, ushort ySize,
        BarCodeType type, ushort mod,
        FontID font = 0,
        MODCIColors color = NEUTRAL_WHITE,
        ushort moduleWidth = 0xFF,
        ushort elementHeight = 0xFFFF,
        ushort heightMultiplier = 1,
        ushort wideNarrowRatio = 0xFFFF);
BCOCA(const BCOCA& bcoca);
virtual ~BCOCA();
BCOCA& operator=(const BCOCA& bcoca);
int operator==(const BCOCA&) const;

```

The default constructor takes only one parameter, which is the name of the BCOCA object, used on the BOG (Begin Object Environment Group) structured field. The name can be up to 252 characters long. The default constructor builds a 3-of-9 bar code, with no check digit, using an 8.5 x 11 inch presentation space and units of 1440ths of an inch.

There are two long form constructors. The first form also defaults to an 8.5 x 11 inch presentation space and units of 1440ths of an inch. The second form lets you specify the unit base, units per unit base, and the size of the presentation space. Both forms take the following parameters:

**Bar Code Type** specifies the type of bar code symbol generated. Many subsequent parameters are dependent on the bar code type, so be sure you understand how your type of bar codes should be implemented. Valid values are:<sup>5</sup>

```

CODE_39_BC or AIM_USS_39_BC      // AIM USS-39, Code 39 (3 of 9)
MSI_BC                          // Modified Plessey
UPC_A_BC or UPC_CGPC_VER_A_BC   // UPC/CGPC Version A
UPC_E_BC or UPC_CGPC_VER_E_BC   // UPC/CGPC Version B
UPC_2_DIGIT_SUPP_BC             // UPC - two digit supplemental
UPC_5_DIGIT_SUPP_BC             // UPC - five digit supplemental
EAN_8_BC                        // EAN-8 (includes JAN-short)
EAN_13_BC                       // EAN-13 (includes JAN-standard)
INDUSTRIAL_2_OF_5_BC            // Industrial 2-of-5
MATRIX_2_OF_5_BC                // Matrix 2-of-5
AIM_USS_25_BC or INTERLEAVED_2_OF_5_BC
                                // Interleaved 2-of-5, AIM USS-I 2/5
CODABAR_BC or CODABAR_2_OF_7_BC or AIM_USS_CODABAR_BC
                                // AIM USS-Codabar, Codabar 2-of-7
CODE_128_BC or AIM_USS_128_BC   // CODE 128, AIM USS-128
EAN_2_DIGIT_SUPP_BC             // EAN Two-digit supplemental
EAN_5_DIGIT_SUPP_BC             // EAN Five-digit supplemental
POSTNET_BC                      // POSTNET

```

**Bar Code Modifier** gives additional processing information about the bar code symbol generated. For example, it indicates whether a check-digit is generated for the bar code symbol. The meaning of the modifier values will vary depending on the type of bar code symbol. See the *Bar Code Object Content Architecture Reference* for a complete description of modifier meanings. Valid values by type are:

---

5. The abbreviations used in these descriptions have the following meanings:

AIM USS: Automatic Identification Manufacturers Uniform Symbol Specification  
 EAN: European Article Numbering  
 JAN: Japanese Article Numbering  
 MSI: MSI Data Corporation  
 POSTNET: POSTal Numeric Encoding Technique (U.S. Postal Service)  
 UPC: Universal Product Code (United States)  
 UPC/CGPC: Universal Product Code (United States) and the Canadian Grocery Product Code

## BCOCA Class

CODE_39_BC or AIM_USS_39_BC	x'01' and x'02'
MSI_BC	x'01' through x'09'
UPC_A_BC or UPC_CGPC_VER_A_BC	x'00'
UPC_E_BC or UPC_CGPC_VER_E_BC	x'00'
UPC_2_DIGIT_SUPP_BC	x'00'
UPC_5_DIGIT_SUPP_BC	x'00'
EAN_8_BC	x'00'
EAN_13_BC	x'00'
INDUSTRIAL_2_OF_5_BC	x'01' and x'02'
MATRIX_2_OF_5_BC	x'01' and x'02'
AIM_USS_25_BC or INTERLEAVED_2_OF_5_BC	x'01' and x'02'
CODABAR_BC or CODABAR_2_OF_7_BC or AIM_USS_CODABAR_BC	x'01' and x'02'
CODE_128_BC or AIM_USS_128_BC	x'02'
EAN_2_DIGIT_SUPP_BC	x'00'
EAN_5_DIGIT_SUPP_BC	x'00'
POSTNET_BC	x'00' through x'03'

**Font** is ignored and the value of X'FF' (device default) is always used. This parameter is provided as a place holder for possible future enhancements. See the *Bar Code Object Content Architecture Reference* for a description of the fonts and code pages required by the various bar code types.

**Bar Code Color** specifies the color in which the bars of the symbol and the HRI data are presented. Valid values are:

BLUE  
RED  
MAGENTA  
GREEN  
CYAN  
YELLOW  
BLACK  
BROWN  
MEDIUM\_COLOR  
DEFAULT\_COLOR

**Module Width** specifies the width in mils (thousandths of an inch) of the smallest defined bar code element. Some bar code symbologies refer to this value as the unit or X-dimension width. The widths of all symbol elements (bars and spaces) are normally expressed as multiples of the module width. Specify -1 to use the default.

**Element Height** specifies the height of the bar code symbol in units of 1440ths of an inch. The element height and height multiplier values are used to compute the total bar and space height of the bar code symbol. Valid values are 0x0000 through 0x7FFF, and 0xFFFF which indicates that the default element height of the presentation device is used.

**Height Multiplier** specifies a value that, when multiplied by the element height, yields the total bar and space height presented. Valid values are 1 through 255.

**Wide-to-narrow Ratio** specifies the ratio of the wide-element dimension to the narrow-element dimension for a two level bar code symbol. The value is given as a real decimal number and normally varies between 2.00 and 3.00. You can specify a value of -1 to indicate that the device default should be used. This parameter is ignored for POSTNET, EAN, and UPC type bar codes.

## Public Members of BCOCA

### AddBarCode

```
void AddBarCode(ushort xPos, ushort yPos, const AString& data,
                boolean hri = TRUE, boolean hriOnTop = FALSE,
                boolean hriOnBottom = FALSE, boolean asterisk = FALSE);
```

The AddBarCode function is used to place the bar code data in the bar code object. You can control the placement of data, the information encoded using parameters specified on the BarCode constructor, and the formatting of the HRI.

The **xPos** and **yPos** variables specify the offset of the bar code symbol origin from the presentation space origin (see Figure 7 on page 33). The bar code must be placed such that the entire bar code, including any HRI, fits within the defined presentation space. If not, an error will occur at print time.

The **data** parameter contains the data encoded in the bar code. The format and length of the data is dependent on the type of bar code being generated. For example, UPC Version A bar codes require a



string of exactly 11 digits, while Code 39 bar codes can have strings of up to 50 alphanumeric and special characters. The output code page for the bar code data depends on the bar code type, and is normally EBCDIC code page 500 or 893. The input code page is controlled with the **AFPDoc::SetInputCodepage** function (or the default of code page 850).

The **hri** flag indicates whether the text for the bar code (HRI) will be presented with the bar code data. The **hriOnTop** and **hriOnBottom** flags control the placement of the HRI. If both flags are FALSE, or both flags are TRUE, the text will be placed in the default position as defined by the bar code type. These flags are ignored if **hri** is FALSE.

The **asterisk** flag is valid only for Code 39 bar codes, and specifies whether an asterisk is presented as the text for the Code 39 start and stop characters. This parameter is ignored for all other bar code types.

### CopyOca

```
virtual OCA* CopyOca(void) const;
```

Returns a properly typed copy of this object. All subclasses of BCOCA must reimplement this function.

### Export

```
ostream& Export(ostream& os) const;
```

Outputs the BCOCA object to the specified output stream. The stream specified must have been previously opened for binary output.

### Font

```
FontID Font(void) const;
```

Returns the HRI Font ID for this bar code object.

### Mnemonic

```
AEString Mnemonic(void) const;
```

Returns the string "BCOCA" to describe this object.

# Graphic Object Content Architecture (GOCA)

The Graphic Object Content Architecture (GOCA) object class encapsulates an AFP graphic object. Graphic objects are used to represent pictures generated by computer. Typically, pictures are built from many different kinds of primitives, such as:

- lines or arcs
- characters or symbols
- shaded areas or boxes

See the *Graphics Object Content Architecture Reference* for more information about GOCA.

## Declarations for the GOCA Class

To use the GOCA class functions, specify the following in your program:

```
#define INCL_GOCA
#define INCL_GOCASF
#include (afptk.h)
```

These statements cause AFP Toolbox to include the appropriate header files at compile time. The following is an excerpt from the **goca.h** file that shows the relevant declarations for the members of GOCA.

```
class AOCL_CLASS GOCA: public OCA, public SFContainerUnsorted
{
    friend ostream& operator<<(ostream&, const GOCA&);

public:

    GOCA(const AESTring& GOCAName = emptyStr);
    GOCA(const AESTring& GOCAName, short xlw, short xrw,
          short ybw, short ytw);

    GOCA(const GOCA& goca);
    virtual ~GOCA();

    int operator==(const GOCA&) const;
    GOCA& operator=(const GOCA& goca);
    virtual OCA* CopyOCA(void) const;

    void BeginArea(boolean drawbound);
    void BeginImage(ushort width, ushort height);
    void CharString(const AESTring& GOCAStr);
    void DrawBox(short cornerXPos, short cornerYPos,
                 ushort haxisval, ushort vaxisval);
    void DrawFillet(short count, short* xypairarray);
    void DrawMarkers(short count, short* xypairarray);
    void DrawRules(short count, short* xypairarray);
    void EndArea(short datalen);
    void EndImage(short datalen);
    void EndProlog(void);
    void FullArc(byte intmult, byte fractmult);
    void ImageData(short imlen, byte* imdata);
    void NoOP(void);
    void QueryGPSPos(short* XPos, short* YPos);
    void RelativeRules(short count, short* xypairarray);
    void SetArcParms(short xmaj, short ymin, short xmin, short ymaj);
    void SetBackgroundMix(byte mode);
    void SetCharAngle(short xpos, short ypos);
    void SetCharCell(short cellwi, short cellhi,
                     ushort cellwfr, ushort cellhfr);
    void SetCharDir(byte chadir);
    void SetCharPrec(byte precision);
    void SetCharSet(ushort fontid);
    void SetColor(byte color);
    void SetGPSPos(short newXPos, short newYPos);
    void SetExtColor(MODCAColors colorval);
```

```

void SetLineType(byte type);
void SetLineWidth(byte width);
void SetMarkerPrec(byte precision);
void SetMarkerSym(byte code);
void SetMix(byte order);
void SetPatternSym(byte pattern);
void SetProcessColor(byte colspce, uint colsize1, uint colsize2,
                    uint colsize3, uint colsize4, ulong colval1,
                    ulong colval2, ulong colval3, ulong colval4);

```

## Constructors and Operators

```

GOCA(const AString& GOCAName = emptyStr);
GOCA(const AString& GOCAName, short xlw, short xrw,
      short ybw, short ytw);

GOCA(const GOCA& goca);
virtual ~GOCA();

int operator==(const GOCA&) const;
GOCA& operator=(const GOCA& goca);

```

The default constructor takes only one parameter, which is the name of the GOCA object, used on the BGR (Begin Graphic) structured field.

The long form constructor can be used to create a GOCA object where you also specify the graphic window coordinates: The **xlw**, **xrw**, **ybs**, and **ytw** parameters give the left, right, bottom, and top edges respectively of the Graphics Presentation Space (GPS). Keep in mind that the GPS is a Cartesian coordinate system with the origin at (0,0) in the center, and not at the upper left hand corner like an APF page. See Figure 26 for a diagram.

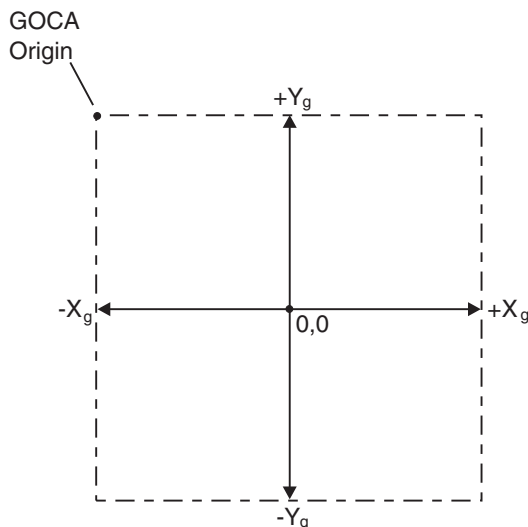


Figure 26. GOCA Presentation Space

## Public Members of GOCA

### BeginArea

```
void BeginArea(boolean drawbound);
```

Begins an area inside a GOCA object. Areas are two-dimensional primitives which are filled with a pattern. The only GOCA orders that are allowed inside an area are:

- Comment
- Draw Box

## GOCA Class

- Fillet
- Full Arc
- Line
- No-op
- Relative Line
- Set Arc Parameters
- Set Extended Color
- Set GPS Position
- Set Line Parm

The Toolbox does not return an error if you try to use other orders inside your graphic object, but the printer will.

This drawing order does not change the current GPS position.

**boolean** specifies whether the border of the area is drawn.

## CharString

```
void CharString(const AString& GOCAStr);
```

Place a string of text in a GOCA object. The string will be formatted in the last font specified with **AFPGSetFont (GOCA Set Font)**. The string is always left-aligned at the current GPS position. The string will be translated to the code page specified on the font definition.

This drawing order does not change the current GPS position.

**Note:** Some older printers do not handle GOCA text correctly. If the text does not format as you think it should, you might want to try placing the character string in your page using **AFPWrite** instead of placing it inside the GOCA object.

## DrawBox

```
void DrawBox(short cornerXPos,short cornerYPos,  
             ushort haxisval,ushort vaxisval);
```

Draw a box using the current GPS position as one corner and the specified X and Y coordinates as the other. The box can have square or rounded corners.

The axis values specify the axes of a full ellipse. A quarter of the ellipse is used to draw each corner of the box. If the horizontal and vertical values are equal, quadrants of a circle are used instead of an ellipse. If both values are zero then a square-cornered box is drawn.

The box will be filled with a pattern if it is inside a graphics area and you have specified a fill pattern with **GOCA Set Pattern Symbol**.

This drawing order does not change the current GPS position.

**Note:** This drawing order is not supported on all IBM printers.

## DrawFillet

```
void DrawFillet(short count, short* xypairarray);
```

This function is used to draw a series of fillets from the current location. The positions for the fillets are supplied as an array of integers, consisting of the X position followed by the Y position. For example, array

position 0 would contain the X coordinate and array position 1 would contain the Y coordinate for the first fillet, array positions 2 and 3 would contain the X and Y coordinate for the second fillet, and so forth.

The current GPS position will be set to the last of the (X,Y) coordinate pairs that are specified.

### DrawMarkers

```
void DrawMarkers(short count, short* xypairarray);
```

This function is used to draw a series of markers at the current and specified locations. The positions for the markers are supplied as an array of integers, consisting of the X position followed by the Y position. For example, array position 0 would contain the X coordinate and array position 1 would contain the Y coordinate for the first marker, array positions 2 and 3 would contain the X and Y coordinate for the second marker, and so forth.

The current GPS position will be set to the last of the (X,Y) coordinate pairs that are specified.

### DrawRules

```
void DrawRules(short count, short* xypairarray);
```

This function is used to draw a series of rules from the current location. The positions for the rules are supplied as an array of integers, consisting of the X position followed by the Y position. For example, array position 0 would contain the X coordinate and array position 1 would contain the Y coordinate for the first rule, array positions 2 and 3 would contain the X and Y coordinate for the second rule, and so forth.

The current GPS position will be set to the last of the (X,Y) coordinate pairs that are specified.

### EndArea

```
void EndArea(short datalen);
```

Ends an area inside a GOCA object. See Begin Area for more information.

This drawing order does not change the current GPS position.

### FullArc

```
void FullArc(byte intmult, byte fractmult);
```

Draw a circle or ellipse at the current GPS position. The arc will be filled with a pattern if it is inside a graphics area and you have specified a fill pattern with GOCA Set Pattern Symbol. The size and shape of the arc are determined by the SetArcParms method.

This drawing order does not change the current GPS position.

### NoOP

```
void NoOP(void);
```

Generates a single byte of zeros in the graphics object. This drawing order is ignored by the printer.

### QueryGPSPos

```
void QueryGPSPos(short* XPos, short* YPos);
```

Queries the current GPS X and Y coordinates as determined by the last Set GPS Position or drawing order that caused the position to move.

### RelativeRules

```
void RelativeRules(short count, short* xypairarray);
```

## GOCA Class

This function is similar to DrawRules, except the X and Y coordinates are given as values relative to the last position rather than absolute.

### SetArcParms

```
void SetArcParms(short xmaj,short ymin,short xmin,short ymaj);
```

Specify the endpoints of the major and minor axes of the ellipse or circle that are drawn. If you want a circle, then xmin and ymaj should be zero.

### SetCharAngle

```
void SetCharAngle(short xpos,short ypos);
```

The character angle controls the angle (measured counter-clockwise) of the character baseline with respect to the GPS X axis. This is similar to the character rotation function in presentation text (**AFPSetTextOrientation**). In AFP printers only angles of 0, 90, 180, and 270 degrees are supported. Select the angle you want by setting X and Y appropriately:

- If X position is positive and Y position is zero, the character angle is zero degrees.
- If X position is zero and Y position is positive, the character angle is 90 degrees.
- If X position is negative and Y position is zero, the character angle is 180 degrees.
- If X position is zero and Y position is negative, the character angle is 270 degrees.

### SetCharCell

```
void SetCharCell(short cellwi,short cellhi,  
                ushort cellwfr,ushort cellhfr);
```

These four parameters set the value of the current character cell-size, used to scale characters specified in subsequent Graphics Character String drawing orders. You can specify both an integer and fractional portion for the width and height values.

**Note:** You might achieve better results by using an outline font and scaling it on the Define Font command.

### SetCharDir

```
void SetCharDir(byte chardir);
```

This parameter controls the placement of a character relative to the previous character. Valid values are:

- DEFAULT\_CD means to use the drawing default, which in AFP environments is the same as LFT2RGHT.
- LFT2RGHT\_CD means that character boxes are positioned from left to right along the GPS X axis.
- TOP2BTM\_CD means that character boxes are positioned from top to bottom along the GPS X axis.
- RGHT2LFT\_CD means that character boxes are positioned from right to left along the GPS X axis.
- BTM2TOP\_CD means that character boxes are positioned from bottom to top along the GPS X axis.

### SetCharPrec

```
void SetCharPrec(byte precision);
```

This parameter specifies how the characters are drawn.

- DEFAULT\_PR means to use the drawing default. In AFP printers this is the same as CHARACTER precision.
- STRING\_PR means to draw the characters using the simplest mode possible for the implementation. Some character attributes might be ignored as a trade-off for improved performance.
- CHARACTER\_PR means to draw the characters taking into account all the attributes to determine the positioning of the characters.

**SetCharSet**

```
void SetCharSet(ushort fontid);
```

Set the current font for GOCA text only. This command does not effect text in the page.

**SetGPSPos**

```
void SetGPSPos(short newXPos, short newYPos);
```

Set the current X and Y coordinates in the Graphics Presentation Space.

**SetExtColor**

```
void SetExtColor(MODCAColors colorval);
```

Set the foreground color attributes for characters, image, lines, markers, and patterns. MODCA Color is any of the defined MODCA color values: BLUE, RED, MAGENTA, GREEN, CYAN, YELLOW, BLACK, or BROWN.

**SetLineType**

```
void SetLineType(byte type);
```

Specifies the pattern used for drawing lines:

- DEFAULT\_LT for a solid line.
- DOTTED\_LT for a dotted line.
- SHORTDSH\_LT for a short dashed line.
- DASHDOT\_LT for a dash-dot line.
- DBLDOT\_LT for a double dotted line.
- LONGDSH\_LT for a long dashed line.
- DSHDBDOT\_LT for a dash-double-dot line.
- SOLID\_LT for a solid line.
- INVSBLE\_LT for an invisible line.

**SetLineWidth**

```
void SetLineWidth(byte width);
```

Width as an integer multiplier of the normal line width (which is approximately .01 inches).

**SetMarkerPrec**

```
void SetMarkerPrec(byte precision);
```

This parameter specifies how the markers are drawn.

- DEFAULT\_PR means to use the drawing default. In AFP printers this is the same as CHARACTER precision.
- STRING\_PR means to draw the markers using the simplest mode possible for the implementation. Positioning might be approximate.
- CHARACTER\_PR means to draw the markers taking into account all the attributes to determine the positioning of the markers.

**SetMarkerSym**

```
void SetMarkerSym(byte code);
```

Specifies the symbol drawn as a marker:

- DEFAULT\_MS is the same as a cross
- CROSS\_MS draws an X

## GOCA Class

- PLUS\_MS draws a plus sign
- DIAMOND\_MS draws a hollow diamond
- SQUARE\_MS draws a hollow square
- SIXSTAR\_MS draws a 6-point star
- EIGHTSTR\_MS draws a 8-point star
- FILLDMND\_MS draws a filled diamond
- FILLSQR\_MS draws a filled square
- DOT\_MS draws a solid dot
- CIRCLE\_MS draws a circle.

## SetPatternSym

```
void SetPatternSym(byte pattern);
```

Specifies the type of fill pattern to use for boxes, circles, ellipses, and other paths that are drawn inside of graphic areas. See Figure 23 on page 299 for samples.

- DEFAULT\_FP is solid fill
- DOT01\_FP through DOT08\_FP or DOT08-FP are dotted patterns of decreasing density
- VERTLN\_FP is vertical lines
- HORZLN\_FP is horizontal lines
- BLTR1\_FP and BLTR2\_FP are diagonal lines from bottom-left to top-right
- TLBR1\_FP and TLBR2\_FP are diagonal lines from top-left to bottom-right
- NOFILL\_FP means no fill pattern
- SOLID\_FP is solid fill.

## SetProcessColor

```
int AOCL_FUNCTION AFGSetProcessColor(TBHANDLE in_gocaHandle, ColorSpace in_coltype,  
                                     MODCAColors in_color, ulong in_colspec1,  
                                     ulong in_colspec2, ulong in_colspec3, ulong in_colspec4);
```

Specifies a process color, highlight color, or named color used for characters, image, lines, markers, and patterns.

**Note:** Use of this drawing order requires microcode support in your printer.

Color Space Type specifies the type of color generated:

- RGB\_COLOR means that the color value is specified as three intensity levels for the red, green, and blue components of the color. These numbers are given for color specification values 1, 2, and 3. The fourth specification is ignored and should be specified as 0.
- CMYK\_COLOR means that the color value is specified as four intensity levels for the cyan, magenta, yellow, and black components of the color. These are color specifications 1-4.
- HILIGHT\_COLOR defines a request for presentation device to generate a highlight color. Specify a MODCA Color value for the highlight color. Color specification 1 is the percentage coverage for the specified color, and color specification 2 is the percentage of black that is added to the coverage (can be 0). Color specifications 3 and 4 should be 0.
- CIELAB\_COLOR is specified with three components. Color specification 1 is the luminance, and color specifications 2 and 3 are the chrominance differences. Color specification 4 should be 0.
- OCA\_COLOR defines a request for presentation device to generate a highlight color. Specify a MODCA Color value for the highlight color. The presentation device will produce printed colors using a color mapping table (see PSF documentation for your environment). Color specifications 1-4 should be 0.



MODCA Color is any of the defined MODCA color values: BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, OR GRAY. This value is only used for Highlight or OCA color spaces but always must be specified due to C language conventions.

Color values 1-4 usage depends on the color space type, see descriptions above.



---

## Chapter 6. Defining Fonts with C++ Language Programs

To define fonts with C++ language programs you must ensure that the fonts you want to define are installed on your system and that you have access to them. To find out what fonts are installed on your system, run whichever program is available in your environment for listing fonts. See the appendix for your operating system for more information.

# Define Double-Byte Font By Attribute

## Function

Locates a font on your system that matches specified character set and code page combinations and returns a font ID. The code page information is used to translate a DBCS string into the font's code set. The function locates only an outline font. An AFP SBCS code page and AFP DBCS code page must be specified. The font ID is used on subsequent Set Font calls.

**Note:** Use of the API requires the access to AFP DBCS outline fonts. (AFP DBCS raster fonts are not supported.) Refer to the AFP Font Collection products for information about how to build DBCS outline fonts. This is the only call for defining a DBCS font.

## Syntax

```
int AFPDefinedbFontByAttr(  
    char*      in_SBCS_codePage,  
    char*      in_DBCS_codePage,  
    char*      in_codeset,  
    char*      in_descriptiveName,  
    ushort    in_pointSize,  
    ushort    in_horizontal_scale_factor  
    FDSWeights in_weight,  
    FDSWidths  in_width,  
    FontStyle  in_style,  
    FontID*    out_fontID  
);
```

## Input Parameters

### SBCS code page

The SBCS code page name for the output data (for example, T1H00290).

### DBCS code page

The DBCS code page name for the output data (for example, T10300).

### Codeset

The code set name for the output data (for example, "IBM-939").

### Descriptive name

Name of the font (for example, "HeiseiMincho"). The value for this parameter is case sensitive.

### Point size

Size of the font specified in decipoints (point size multiplied by 10). For example, a 12 point font would be specified as 120.

### Horizontal scaling factor

A horizontal scaling factor expressed as a percentage of the point size. For example, if you choose a value of 50, the width of your font will be 50% of the height. No scaling is performed if values of either 0 or 100 are entered.

### Weight

The thickness of the font, for example medium or bold. Valid values are:

```
ULTRA_LIGHT_WT  
EXTRA_LIGHT_WT  
LIGHT_WT  
SEMI_LIGHT_WT  
MEDIUM_WT  
SEMI_BOLD_WT  
BOLD_WT  
EXTRA_BOLD_WT  
ULTRA_BOLD_WT
```

**Width**

The width of the font, for example normal or condensed. Valid values are:

```
ULTRA_CONDENSED_WD
EXTRA_CONDENSED_WD
CONDENSED_WD
SEMI_CONDENSED_WD
MEDIUM_WD
SEMI_EXPANDED_WD
EXPANDED_WD
EXTRA_EXPANDED_WD
ULTRA_EXPANDED_WD
```

**Style**

The style of the font, either roman or italic. Valid values are:

```
NORMAL_FS          (roman)
ITALIC_FS
```

**Output Parameters****Font ID**

The font ID for use on subsequent PText::SetFont calls.

**Return Codes**

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

**Sample Function Call**

This example defines a 14 point Heisei Gothic font.

```
rc = AFPDefineDbFontByAttr (
    "T1H01027",          // SBCS codepage
    "T10300",            // DBCS codepage
    "IBM-939",           // Codeset
    "HeiseiKakuGothic",  // Font name
    140,                 // Deci-point size
    0,                   // Scale factor
    MEDIUM_WT,          // Weight
    MEDIUM_WD,          // Width
    NORMAL_FS,           // Font style
    &FontID              // Font handle
);
```

# Define Font By Attribute

## Function

Locates a font on your system that matches specified attributes and returns a font ID corresponding to this font. The font ID is used on subsequent PText::SetFont calls.

**Note:** Define Font can only *define* an existing font to be used, it does not *create* a font with these attributes.

For information about obtaining a list of fonts available on your system, see the appendix for your operating system.

## Syntax

```
int AFPDefineFontByAttr(  
    char*          in_codePage,  
    char*          in_descriptiveName,  
    ushort         in_pointSize,  
    FDSWeights     in_weight,  
    FDSWidths      in_width,  
    FontStyle      in_style,  
    FontID*        out_fontID  
);
```

## Input Parameters

### Code page

The code page name (the member name in the font library, for example, T1V10500) for the output data. This is not necessarily the same as the input code page in which the text data is entered. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, \_ (underscore), #, and @. Code page names must be in uppercase.

The default input code page for all text in AFP Toolbox is assumed to be the multinational ASCII code page T1000850 for workstation systems and EBCDIC T1V10500 for host systems (unless you have changed this value with the AFPSetInputCodepage call).

### Descriptive name

Name of the font (for example, "TIMES NEW ROMAN LATIN1"). The value for the descriptive name parameter is case sensitive. Descriptive names in IBM-supplied fonts are usually defined in uppercase.

### Point size

Size of the font specified in decipoints (point size multiplied by 10). For example, a 12 point font would be specified as 120. For raster fonts, specify a point size you know exists in your font library. For outline fonts, fractional point sizes are allowed. For example, a 12.5 point font would be specified as 125.

### Weight

The thickness of the font, for example medium or bold. Valid values are:

```
ULTRA_LIGHT_WT  
EXTRA_LIGHT_WT  
LIGHT_WT  
SEMI_LIGHT_WT  
MEDIUM_WT  
SEMI_BOLD_WT  
BOLD_WT  
EXTRA_BOLD_WT  
ULTRA_BOLD_WT
```

### Width

The width of the font, for example normal or condensed. Valid values are:

```
ULTRA_CONDENSED_WD  
EXTRA_CONDENSED_WD  
CONDENSED_WD  
SEMI_CONDENSED_WD  
MEDIUM_WD  
SEMI_EXPANDED_WD  
EXPANDED_WD  
EXTRA_EXPANDED_WD  
ULTRA_EXPANDED_WD
```

### Style

The style of the font, either roman or italic. Valid values are:

```
NORMAL_FS          (roman)  
ITALIC_FS
```

## Output Parameters

### Font ID

The font ID for use on subsequent PText::SetFont calls.

## Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

## Sample Function Call

To define a Times New Roman 12 point, bold italic font enter:

```
strcpy(Typeface,"TIMES NEW ROMAN LATIN1");  
Codepage="T1V10500"; /*EBCDIC code page 500*/  
rc = AFPDefineFontByAttr(Codepage, Typeface, 120, BOLD_WT,  
                          MEDIUM_WD, ITALIC_FS, &FontID);
```

# Define Font By Attribute With Scaling

## Function

Locates a font on your system that matches specified attributes and returns a font ID corresponding to this font. The font ID is used on subsequent AFPSetFont calls. To use this call, you must first select an outline font technology with Set Font Type.

This call is only needed if you are using outline fonts that are not evenly scaled in the horizontal and vertical directions.

**Note:** Define Font can only *define* an existing font to be used, it does not *create* a font with these attributes.

For information about obtaining a list of fonts available on your system, see the appendix for your operating system.

## Syntax

```
int AFPDefineFontByAttrWithScaling(  
    char*          in_codePage,  
    char*          in_descriptiveName,  
    ushort         in_pointSize,  
    ushort         in_Hscale  
    FDSWeights     in_weight,  
    FDSWidths      in_width,  
    FontStyle      in_style,  
    FontID*        out_fontID  
);
```

## Input Parameters

### Code page

The code page name (the member name in the font library, for example, T1V10500) for the output data. This is not necessarily the same as the input code page in which the text data is entered. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, \_ (underscore), #, and @. Code page names must be in uppercase.

The default input code page for all text in AFP Toolbox is assumed to be the multinational ASCII code page T1000850 for workstation systems and EBCDIC T1V10500 for host systems (unless you have changed this value with the AFPSetInputCodepage call).

### Descriptive name

Name of the font (for example, "TIMES NEW ROMAN LATIN1"). The value for the descriptive name parameter is case sensitive. All descriptive names in IBM-supplied fonts are usually defined in uppercase.

### Point size

Size of the font specified in decipoints (point size multiplied by 10). For example, a 12 point font would be specified as 120. For raster fonts, specify a point size you know exists in your font library. For outline fonts, fractional point sizes are allowed. For example, a 12.5 point font would be specified as 125.

### H scale

A horizontal scaling factor expressed as a percentage of the point size. For example, if you choose a value of 50, the width of your font will be 50% of the height. No scaling is performed if values of either 0 or 100 are entered.

### Weight

The thickness of the font, for example medium or bold. Valid values are:



ULTRA\_LIGHT\_WT  
EXTRA\_LIGHT\_WT  
LIGHT\_WT  
SEMI\_LIGHT\_WT  
MEDIUM\_WT  
SEMI\_BOLD\_WT  
BOLD\_WT  
EXTRA\_BOLD\_WT  
ULTRA\_BOLD\_WT

### Width

The width of the font, for example normal or condensed. Valid values are:

ULTRA\_CONDENSED\_WD  
EXTRA\_CONDENSED\_WD  
CONDENSED\_WD  
SEMI\_CONDENSED\_WD  
MEDIUM\_WD  
SEMI\_EXPANDED\_WD  
EXPANDED\_WD  
EXTRA\_EXPANDED\_WD  
ULTRA\_EXPANDED\_WD

### Style

The style of the font, either roman or italic. Valid values are:

NORMAL\_FS                    (roman)  
ITALIC\_FS

## Output Parameters

### Font ID

The font ID for use on subsequent Set Font calls.

## Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

## Sample Function Call

To define a Times New Roman 12 point, bold italic font scaled horizontally to 50 percent of the point size enter:

```
strcpy(Typeface,"TIMES NEW ROMAN LATIN1");  
Codepage="T1V10500"; /*EBCDIC code page 500*/  
rc = AFPSetFontType(FONT_OLN);  
rc = AFPDefineFontByAttrWithScaling(Codepage, Typeface, 120, 50, BOLD_WT,  
                                     MEDIUM_WD, ITALIC_FS, &FontID);
```

## Define Font By Name

### Function

Locates a raster technology font on your system that matches the specified character set and code page combination and returns a font ID. The font ID is used on subsequent SetFont calls.

For information about obtaining a list of fonts available on your system, see the appendix for your operating system.

### Syntax

```
int AFPDefineFontByName(  
    char*    in_codePage,  
    char*    in_characterSet,  
    char*    in_codedFont,  
    FontID   out_fontID  
);
```

### Input Parameters

#### Code page

The code page name (the member name in the font library, for example, T1V10500) for the output data. This is not necessarily the same as the input code page in which the text data is entered. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, \_ (underscore), #, and @. Code page names must be in uppercase.

The default input code page for all text in AFP Toolbox is assumed to be the multinational ASCII code page T1000850 which is available for all workstation systems and EBCDIC T1V10500 for host systems (unless you've changed this value with the Set Input Codepage call).

#### Character set

The character set name (member name in the font library). For example, C0H20000 for the Courier 10 point Latin characters. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, \_ (underscore), #, and @. Character set names must be in uppercase.

#### Coded font

The coded font name (member name in the font library). This parameter is currently unsupported, it must be set to NULL.

### Output Parameters

#### Font ID

The font ID for use on subsequent PText::SetFont calls.

### Return Codes

See "Font Return Codes" on page 259 for a listing of the return codes for this call.

### Sample Function Call

Define a font given a character set and code page combination:

```
CharSet="C0N20000"; /*Courier 10 point normal character set*/  
Codepage="T1V10500"; /*EBCDIC codepage 500*/  
rc = AFPDefineFontByName(Codepage, CharSet, NULL, &FontID);
```

## Measure Double-Byte String

### Function

Measures the width of the specified string in the given font and returns the width. The specified font must be available to the AFP Toolbox or this call fails. The Measure Double-Byte String function assumes that the input DBCS character string is encoded in the code set of the specified font. To use a different input code set, see “Translate And Measure Double-Byte String” on page 368. The string can contain both SBCS and DBCS characters.

**Note:** You do not need to measure every string that is placed in your document. Measuring a lot of strings has a negative impact on the performance of your program.

### Syntax

```
int AFPMeasureDbString(
    FontID      in_fontID,
    char*       in_inputString,
    long*       outp_stringWidth
    ushort     in_vinc,
    ushort     in_cadj,
    int*        outp_varspaces
    int*        outp_icspaces,
);
```

### Input Parameters

#### Font ID

The font ID returned by the Define Double-Byte Font By Attribute call.

#### Character string

The string of DBCS characters to be measured. The text is assumed to be encoded in the code set of the defined font. The defined font must be located with Define Double-Byte Font With Attribute.

#### Variable increment

The width of the space between words in the specified font measured in 1440ths of an inch. The default value is defined by the font. Specify 0 to use the default value.

#### Extra intercharacter space

Amount of extra space to be added to each intercharacter space (measured in 1440ths of an inch) as the string is measured. Specify 0 to use the default value.

### Output Parameters

#### String width

The width of the string in the specified font. String width is returned in units of 1440 per inch.

#### Variable spaces

A count of the number of spaces between words in the line of text. If this value is returned as 0 and there are spaces in your input string, ensure that your input string is encoded in the code page of the specified font. The encoding of a blank varies based on whether the code page is ASCII or EBCDIC.

#### Intercharacter spaces

A count of the number of character spaces in the line of text.

### Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

## Measure Double-Byte String

### Sample Function Call

This example assumes you have already received the FontID from the Define Double-Byte Font By Attribute call.

```
int ic_sp, var_sp, rc;
char CharString[LENGTH];
long StringWidth;
strcpy(CharString, "SBCS<DBCS>SBCS");
    where <,> are SOSI in host codepages
        SBCS is a string of SBCS
        DBCS is a string of DBCS

rc = AFPMeasureDbString (FontID,
                        CharString,
                        0,
                        0,
                        &StringWidth,;
                        &ic_sp,;
                        &var_sp);
```

## Measure String

### Function

Measures the width of the specified string in the given font and returns the width. The specified font must be available to AFP Toolbox or this call will fail. The AFPMeasureString function assumes that the input character string is encoded in the code page of the specified font. To use a different input code page, see “Translate And Measure String” on page 370.

You can use the AFPQuery function to obtain the font linespacing value.

**Note:** You does not need to measure every string that is placed in your document. Measuring a lot of strings has a negative impact on the performance of your program.

### Syntax

```
int AFPMeasureString(
    FontID      in_fontID,
    char*       in_inputString,
    long*       outp_stringWidth,
    ushort*     in_vinc,
    ushort*     in_cadj,
    int*        outp_varspaces,
    int*        outp_icspaces,
);
```

### Input Parameters

#### Font ID

The font ID returned by the Define Font By Attributes or Define Font By Name call.

#### Character string

The string of characters to be measured. The text is assumed to be encoded in the code page of the defined font. No translation is performed.

#### Variable increment

The width of the space between words in the specified font measured in 1440ths of an inch. The default value is defined by the font. Specify 0 to use the default value.

#### Extra intercharacter space

Amount of extra space to be added to each intercharacter space (measured in 1440ths of an inch) as the string is measured. Specify 0 to use the default value.

### Output Parameters

#### String width

The width of the string in the specified font. String width is returned in units of 1440 per inch.

#### Variable spaces

A count of the number of spaces between words in the line of text. If this value is returned as 0 and there are spaces in your input string, ensure that your input string is encoded in the code page of the specified font. The encoding of a blank varies based on whether the code page is ASCII or EBCDIC.

#### Intercharacter spaces

A count of the number of character spaces in the line of text.

### Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

## Measure String

### Sample Function Call

This example assumes you have already received the FontID from either the Define Font By Name or Define Font By Attributes call and defined it with the same code page as the default code page for your system.

```
int ic_sp,var_sp;rc;
char CharString[LENGTH];
long StringWidth;

strcpy(CharString,"This is the string to measure");
rc = AFPMeasureString(FontID, CharString, 0, 0, &StringWidth,;
                      &ic_sp, &var_sp);
```

## Set Font Type

### Function

Specifies the font type to use for formatting subsequent text data.

### Syntax

```
int AFPSetFontType(
    FontCSType in_type
);
```

### Input Parameters

#### Type

The font type can be FONT\_240, FONT\_300, or FONT\_OLN. It describes the font technology used for all font definitions in the document. The font type remains active until it is changed with another AFPSetFontType call.

#### OS/2 and AIX Systems

The font type defines the file extension for the font members (for example, .240, .300, or .oln in the font name).

#### MVS and OS/400 Systems

The font type is used to determine which AFP font technology to use. The AFP Toolbox searches all of the font libraries specified with the MVS FNTLIBDD or the OS/400 library list and uses the first member found that matches all of the attributes, including font technology. However, since FONT\_240 and FONT\_300 libraries have the same member name, the AFP Toolbox cannot distinguish between these and will read the first one in the search order.

The default in all environments is FONT\_240.

### Return Codes

- 4 Page handle is not pointing to a valid page. Make sure you have called AFPBgnPage before issuing this call.

### Sample Function Call

This example selects outline font technology with Set Font Type and then defines a font that is 14 points and is scaled horizontally 50 percent.

```
rc = AFPSetFontType(FONT_OLN);
rc = AFPDefineFontByAttrWithScaling(outputCodepage,
    "TIMES NEW ROMAN LATIN1", 140, 50,
    BOLD_WT, MEDIUM_WD, NORMAL_FS,
    &titleFont);
```

# Translate And Measure Double-Byte String

## Function

Translates the string of DBCS characters from **in\_inputCodePage** to the code set requested when **in\_fontID** font was defined, then measures the translated string using the metrics from the specified font.

**Note:** You do not need to measure every string that is placed in your document. Measuring a lot of strings has a negative impact on the performance of your program.

## Syntax

```
AFPXlateAndMeasureDbString(  
    FontID      in_fontID,  
    char*       in_inputString,  
    long*       outp_stringWidth,  
    char*       in_inputCodeSet,  
    ushort      in_vinc,  
    ushort      in_icadj,  
    int*        outp_varspaces,  
    int*        outp_icspaces  
);
```

## Input Parameters

### Font ID

The font ID returned by the Define Double-Byte Font By Attributes call.

### Character string

The string of characters to be translated from **in\_inputCodePage** to the output code page and measured.

### Input code set

Specifies the code set used to encode the input DBCS character string. The defined font must be located with the AFPDefineDbFontWithAttr call.

### Variable increment

The width of the space between words in the specified font measured in 1440ths of an inch. The default value is defined by the font. Specify 0 to use the default value.

### Extra intercharacter space

Amount of extra space to be added to each intercharacter space (measured in 1440ths of an inch) as the string is measured. Specify 0 to use the default value.

## Output Parameters

### String width

The width of the string in the specified font. String width is returned in units of 1440 per inch.

### Variable spaces

A count of the number of spaces between words in the line of text.

### Intercharacter spaces

A count of the number of character spaces in the line of text.

## Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.



## Sample Function Call

Measure the string “hello” in Heisei Gothic font, using input code set IBM-939. Zero is specified for the variable increment and extra intercharacter space parameters, to indicate that the default values should be used. NULL is used as the address of the last two parameters, since we do not care how many characters or spaces were in the string.

```
int rc;
long spwidth;

rc = AFPDefineDbFontByAttr ("T1H01027",      // SBCS codepage
                           "T10300",        // DBCS codepage
                           "IBM-939",       // Codeset
                           "HeiseiKakuGothic", // Font name
                           140,             // Deci-point size
                           0,               // Scale factor
                           MEDIUM_WT,      // Weight
                           MEDIUM_WD,      // Width
                           NORMAL_FS,      // Font style
                           &FontID);        // Font ID

rc = AFPXlateAndMeasureDbString(FontID,      // Font ID
                                "hello",     // DBCS text
                                "IBM-932",   // text's codeset
                                0,           // variable increment
                                0,           // Extrainterchar space
                                &spwidth,    // string width
                                NULL,
                                NULL);
```

# Translate And Measure String

## Function

Translates the string of characters from **in\_inputCodePage** to the code page requested when **in\_fontID** font was defined, then measures the translated string using the metrics from the specified font.

**Note:** You do not need to measure every string that is placed in your document. Measuring a lot of strings has a negative impact on the performance of your program.

## Syntax

```
AFPXlateAndMeasureString(  
    FontID      in_fontID,  
    char*       in_inputString,  
    long*       outp_stringWidth,  
    char*       in_inputCodePage,  
    ushort      in_varinc,  
    ushort      in_icharspace,  
    int*        outp_varsplaces,  
    int*        outp_icspaces  
);
```

## Input Parameters

### Font ID

The font ID returned by the Define Font By Attributes or Define Font By Name call.

### Character string

The string of characters to be translated from **in\_inputCodePage** to the output code page and measured.

### Input code page

Up to 8 characters, specifies the code page used to encode the input character string. If more than 8 are specified, only the first 8 characters are used.

### Variable increment

The width of the space between words in the specified font measured in 1440ths of an inch. The default value is defined by the font. Specify 0 to use the default value.

### Extra intercharacter space

Amount of extra space to be added to each intercharacter space (measured in 1440ths of an inch) as the string is measured. Specify 0 to use the default value.

## Output Parameters

### String width

The width of the string in the specified font. String width is returned in units of 1440 per inch.

### Variable spaces

A count of the number of spaces between words in the line of text.

### Intercharacter spaces

A count of the number of character spaces in the line of text.

## Return Codes

See “Font Return Codes” on page 259 for a listing of the return codes for this call.

## Sample Function Call

Measure the string “hello” in Helvetica 10-point font, using input code page T1000850 and output code page T1V10500. Zero is specified for the variable increment and extra intercharacter space parameters, to indicate that the default values should be used. NULL is used as the address of the last two parameters, since we do not care how many characters or spaces were in the string.

```
FontID textfont; long spwidth;
char inputcp[]="T1000850";
char outputcp[]="T1V10500";
char tempchar[]="hello";
...
AFPDefineFontByAttr(outputcp,"Helvetica Latin1",100,MEDIUM_WT,
                    MEDIUM_WD,NORMAL_FS,&textfont);
AFPXlateAndMeasureString(textfont,tempchar,inputcp,
                        0,0,&spwidth,NULL,NULL);
```

## Translate And Measure String

---

## Part 4. Appendixes



---

## Appendix A. AFP Toolbox and the OS/390 Environment

This appendix contains the following information about using the AFP Toolbox in the MVS or OS/390 environment:

- What prerequisites and system requirements you need
- How to define fonts
- How to compile and run sample programs
- Some troubleshooting information.

### Notes:

1. Throughout this publication, the term MVS applies to both OS/390 and MVS.
2. The sample programs in this publication are presented only as samples, please use the members that are actually installed on your system for testing purposes.

---

## Prerequisites and System Requirements

The IBM AFP Toolbox product for MVS requires one of the following MVS operating systems:

- MVS/ESA™ SP™ Version 4.2 (5695-047 and 5695-048) and above
- MVS/ESA SP Version 5.1 (5655-068 and 5655-069) and above
- OS/390 Release 1 (5645-001) and above

One of the following C/C++ run time Libraries:

- Language Environment® for MVS & VM Version 1.5 (5688-198) or higher
- MVS C/C++ Language Support Feature of MVS/ESA Version 5.2 (5655-068 or 5655-069) or higher
- Language Environment element of OS/390 Version 1 (5645-001).

SMP/E Version 1.8 (5668-949) or higher for installation and maintenance.

The following C/C++ compilers (required for compile time) for C and C++ applications:

- IBM C/C++ for MVS/ESA V3R2 (5655-121) and its prerequisites

If an application other than C or C++ is used to call the AFP Toolbox APIs, the appropriate compiler and run time libraries will also be required. For example, a COBOL program would require the following:

- IBM COBOL for MVS Compiler (5688-197) Version 1 Release 2 or higher and prerequisites

AFP Font Collection Version 2 (5648-B33) or equivalent fonts are also required.

Invoking the AFP Toolbox from CICS® applications is not supported at this time.

---

## Using Fonts with the AFP Toolbox

Before using the AFP Toolbox, you must do the following:

- Install the AFP Font Collection or equivalent AFP font libraries.

AFP Font Collection contains the AFP character set and code page files required by the AFP Toolbox to properly handle converting text from code page to code page and to measure strings of text for exact placement on a page.

Font libraries installed by the AFP Font Collection are required at run time and are specified with the FNTLIBDD DD statement when the application is invoked so that the AFP Toolbox has access to the appropriate font information.

- Your system programmer must run the FLIPMVS utility that is supplied with the AFP Toolbox.

## AFP Toolbox and MVS

The files built by FLIPMVS are specified at run time with the TBXFNTDD DD statement. For more information about running FLIPMVS, see the MVS Toolbox Program Directory.

- Consult the AFP Font Collection documentation to determine which libraries contain the code pages and character sets that your application uses and add these libraries to the DD statement for your application.

**Note:** The AFP Toolbox does not support the use of unbounded box fonts (3800, typically found in SYS1.FONTLIB).

---

## Compiling and Running the Sample Programs

There are sample programs included with the AFP Toolbox for the C, COBOL, and C++ languages. By default, these are installed in the **ATX.SATXSAMP** data set. At least one of these programs should have been run by your system programmer as part of the install verification process. We suggest that you compile and run these sample programs as an exercise to familiarize yourself with the AFP Toolbox product.

The sample JCL shown below assumes that the compilers have been installed in the default libraries and have already been tested. It might be necessary for you to modify the JCL described here to account for differences between the default installation and your configuration. This JCL is installed in the **ATX.SATXJCL(ATXCBSAM)** data set by default.

### COBOL Sample

To compile, link, and run the COBOL language sample program, follow these steps:

**Note:** This JCL assumes that you have access to the IBM supplied IGYCRCTL procedure that comes with the IBM COBOL for MVS compiler.

1. Replace “XXX” with your user ID and modify the job statement as appropriate.
2. Change the TBXFNTDD and FNTLIBDD statements to point to the data sets where your fonts were installed by the system programmer.
3. Modify the FILEOUT DD statement to specify a valid output data set, and execute this job. It produces an AFP document in the data set that you specify. You can print or view this file and compare it with the sample shown in Figure 27 on page 378. To print, you must make the page segments and overlays used in the sample available to PSF, either by copying them to the system AFP resource data sets or by specifying a USERLIB to PSF. By default, the resources are installed in ATX.SATXAFP. The PSF resource data sets must be in VBM format.

```
//XXXTST JOB (TSS40000), 'XXX',MSGLEVEL=(1,1),NOTIFY=XXX,CLASS=A,
//          MSGCLASS=T
//*
/* SAMPLE JCL TO COMPILE, PRELINK, LINK & RUN
/* THE SAMPLE COBOL PROGRAM ATXSSEED, THE
/* SUPER SUN SEEDS EXAMPLE
/*
/*
/*---COMPILE STEP
//COBOL EXEC PGM=IGYCRCTL,REGION=2048K,
//          PARM='OBJECT,LIST,LIB,RENT,PGMNAME(LONGMIXED),NODYNAM'
//STEPLIB DD DSNAME=IGY.V1R2M0.SIGYCOMP,DISP=SHR
//SYSIN DD DSNAME=ATX.SATXSAMP(ATXCBTST),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN DD DSNAME=&&LOADSET,UNIT=SYSDA,
//          DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//          DCB=(BLKSIZE=3200)
/* SYSLIB IS WHERE TO FETCH THE COBOL COPY FILES
//SYSLIB DD DSNAME=ATX.SATXSAMP,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
```



```

//SYSUT4 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//*
//*---PRELINK STEP
//PLKED EXEC PGM=EDCPRLK,COND=(8,LT,COBOL),
// REGION=2048K,
// PARM='MAP,NOER,NONCAL'
//STEPLIB DD DSN=CEE.V1R5M0.SCEERUN,DISP=SHR
//SYSMGS DD DSN=CEE.V1R5M0.SCEMSGP(EDCPMSG),DISP=SHR
//SYSLIB DD DSN=CEE.V1R5M0.SCEECPP,DISP=SHR
//SYSIN DD DSN=ATX.LOADSET,DISP=(OLD,DELETE)
// DD DDNAME=SYSIN2
//SDECK DD DSN=ATX.SATXIMP,DISP=SHR
//SYSMOD DD DSN=ATX.PLKSET,UNIT=VIO,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD DUMMY
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN2 DD *
// INCLUDE SDECK(ATXCOBOL)
// INCLUDE SDECK(ATXTBOX)
// NAME ATXCBTST(R)
//*
//*---LINK STEP
//LKED EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K,
// PARM='LIST,AMODE=31,RMODE=ANY,RENT,CALL'
//SYSLIB DD DSN=CEE.V1R5M0.SCEELKED,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN DD DSN=*.PLKED.SYSMOD,DISP=(OLD,DELETE)
//SYSLMOD DD DSN=ATX.GOSET(GO),SPACE=(TRK,(10,10,1)),
// UNIT=SYSDA,DISP=(NEW,PASS)
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(10,10))
//*
//*---GO STEP
//GO EXEC PGM=*.LKED.SYSLMOD,
// COND=((8,LT,COBOL),(4,LT,LKED)),
// REGION=0K
//STEPLIB DD DSN=CEE.V1R5M0.SCEERUN,DISP=SHR
// DD DSN=ATX.SATXLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//FNTLIBDD DD DSN=SYS1.FONTLIBB,DISP=SHR
//TBXFNTDD DD DSN=SYS1.TBOX.FONTS,DISP=SHR
//DATAFILE DD DSN=ATX.SATXSAMP(SUNDATA),DISP=SHR
//FILEOUT DD DSN=ATX.ATXCBTST.AFP,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(1,1)),
// DCB=(DSORG=PS,RECFM=VBM,LRECL=8205,BLKSIZE=8209)

```

## AFP Toolbox and MVS


400 CPU Parkway Vegetation, NJ 55090		 <b>Super Sun Seeds</b> A Growth Company		Office: 555-499-2367 Fax: 555-415-9794	
<div style="border: 1px solid black; padding: 5px;"> <div style="border-bottom: 1px solid black; margin-bottom: 5px;"> <div style="display: flex; justify-content: space-between;"> <div>           LOS ARBOLES DEL MUNDO            32483 ARBOL LANE            MESA VERDE            IL 65478-9390         </div> <div>           -- Sold To --         </div> <div>           -- Ship To --         </div> </div> </div> </div>					
Customer Number:	141	Invoice Number:	31341	Invoice Date:	7/28/95
				Payment Date:	8/28/95
Ship Via:		Ship Date: 7/28/95		Terms: N10	
				Salesman: MICHELLE	
Qty	UOM	Item #	Item Description	Price	Extension
1000	BX	56413213	POT POT	7.65	7,650.00
45	BZ	11005015	CHANTENAY SEEDS	2.19	98.55
900	EA	00001200	ARBOLES DEL SUR	45.00	40,500.00
98	PK	84512023	OREGON SPRING TOMATO SEED	.97	95.06
4	BX	11057893	AFRICAN DAISY, SEEDS	2.35	9.40
951	CT	11005011	LASSO RED SEEDS	892.23	848,510.73
46	DZ	11005014	SCARLET NANTES SEEDS	5.90	271.40
100	EA	31321655	SEMILLAS DEL SUS SOMBEROS	24.95	2,495.00
<i>Thank you Los Arboles Del Mundo          Because almost half your order was Lasso          Sun Seeds, you will receive a 10%          discount on your next order.</i>					
Total Due					899,630.14

Figure 27. COBOL Sample

## C Sample

A sample job is supplied for C programs, to compile, link, and execute the toolbox application named CTEST. The source for this application is installed in ATX.SATXSAMP(ATXCTEST) by default. The JCL is installed in ATX.SATXJCL(ATXCSAM).

Modify the JCL as follows:

**Note:** This JCL assumes that you installed the IBM C compiler for MVS/ESA V3R2 (5655-121) and its prerequisites, and Language Environment for MVS & VM Version 1.5 (5688-198), in their default libraries.

1. Replace “xxx” with your user ID and modify the job statement as appropriate.
2. If you are using a different compiler or Language Environment, you need to replace the CBC and CEE data set names with your own. If you use your own JCL, be sure and define the MVS and IBMHOST preprocessor macros either in your application with #define statements or with the DEFINE compile option.
3. Change the TBXFNTDD and FNTLIBDD statements to point to the data sets where your fonts were installed by the system programmer.
4. Modify the FILEOUT DD statement to specify a valid output data set, and run this job. It produces an AFP document in the data set that you specify.
5. If you want to print the job, you must copy the AFP resources from ATX.SATXAFP to a VBM dataset accessible to PSF/MVS at print time (such as USERLIB).

To compile, link, and run the C language sample program:

```

//xxxCSAMP JOB (TSS40000),'xxx',MSGLEVEL=(1,1),NOTIFY=xxx,CLASS=A,
//          MSGCLASS=T
//*
//* SAMPLE JCL TO COMPILE, PRELINK, LINK & RUN
//* THE SAMPLE C PROGRAM ATXCTEST, THE
//* WOMBAT SAVINGS AND LOAN EXAMPLE
//*
//*
//*-----
//*  COMPILE STEP:
//*-----
//COMPILE EXEC PGM=CBCCRVR,REGION=0K,
//  PARM=(' /CXX OPTFILE(DD:CCOPT) ')
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
//          DD DSN=CBC.SCBCMP,DISP=SHR
//SYSMSGSGS DD DUMMY,DSN=CBC.SCBCMSG(EDCMSGE),DISP=SHR
//SYSXMSGSGS DD DUMMY,DSN=CBC.SCBCMSG(CBCMSGGE),DISP=SHR
//SYSIN DD DSN=ATX.SATXSAMP(ATXCTEST),DISP=SHR
//SYSLIN DD DSN=*&LOADSET,UNIT=VIO,
//          DISP=(MOD,PASS),SPACE=(512,(50,20)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSCPRT DD SYSOUT=*
//SYSUT1 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT6 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10 DD SYSOUT=*
//SYSUT14 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT15 DD SYSOUT=*
//* COMPILER OPTIONS
//CCOPT DD *
//          SYSPATH( /CEE/SCEEH, /CBC/SCLBH, /ATX/SATXCHDR/DOCUMENT)
//          USERPATH( /ATX/SATXCHDR/DOCUMENT)

```

## AFP Toolbox and MVS

```
DEFINE(IBMHOST)
DEFINE(MVS)
OPT
/*
/*-----
/* * PRE-LINKEDIT STEP:
/*-----
//PLKED EXEC PGM=EDCPRLK,REGION=2048K,COND=(8,LT,COMPILE),
// PARM='MAP,NOER,NONCAL'
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
//SYMSGS DD DSN=CEE.SCEMSGP(EDCPMSG),DISP=SHR
//SYSLIB DD DSN=CEE.SCEECPP,DISP=SHR
//SYSIN DD DSN=*.COMPILE.SYSLIN,DISP=(OLD,DELETE)
// DD DSN=CBC.SCLB3SID(IOSTREAM),DISP=SHR
// DD DSN=CBC.SCLB3SID(APPSUPP),DISP=SHR
// DD DSN=CBC.SCLB3SID(COLLECT),DISP=SHR
// DD DDNAME=SYSIN2
//MYDLLS DD DSN=ATX.SATXTIMP,DISP=SHR
//SYSMOD DD DSN=&&PLKSET,UNIT=VIO,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD DUMMY
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN2 DD *
INCLUDE MYDLLS(ATXTBOX)
NAME ATXCTEST(R)
/*
/*-----
/* * LINKEDIT STEP:
/*-----
//LKED EXEC PGM=HEWL,REGION=1024K,COND=(8,LE,PLKED),
// PARM='AMODE=31,RMODE=ANY,MAP,RENT'
//SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR
//SYSLIN DD DSN=*.PLKED.SYSMOD,DISP=(OLD,DELETE)
//SYSMOD DD DSN=&&GOSET(GO),SPACE=(TRK,(10,10,1)),
// UNIT=SYSDA,DISP=(NEW,PASS)
//SYSUT1 DD UNIT=VIO,SPACE=(32000,(30,30))
//SYSPRINT DD SYSOUT=*
/*
/*-----
/* * GO STEP:
/*-----
//GO EXEC PGM=*.LKED.SYSLMOD,
// REGION=0K,
// COND=((8,LT,COMPILE),(4,LT,LKED))
//STEPLIB DD DSN=ATX.SATXLOAD,DISP=SHR
// DD DSN=CEE.SCEERUN,DISP=SHR
// DD DSN=CBC.SCLB3DLL,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//FNTLIBDD DD DSN=SYS1.FONTLIBB,DISP=SHR
//TBXFNTDD DD DSN=xxx.TBOX.FONTS,DISP=SHR
//CTESTOUT DD DSN=xxx.ATXCTEST.AFP,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(1,1)),
// DCB=(DSORG=PS,RECFM=VBM,LRECL=8205,BLKSIZE=8209)
//CTESTIN DD DSN=ATX.SATXDATA(STMNTDAT),DISP=SHR
```

## C++ Sample

The sample job is supplied for C++ programs, to compile, link, and run the AFP Toolbox application named CTEST. The source for this application is installed in ATX.SATXSAMP(ATXCPTST) by default. The JCL is installed in ATX.SATXJCL(ATXCPSAM).

Modify the JCL as follows:

**Note:** This JCL assumes that you installed the IBM C++ compiler for MVS/ESA V3R2 (5655-121) and its prerequisites, and Language Environment for MVS & VM Version 1.5 (5688-198), in their default libraries.

1. Replace “xxx” with your user ID and modify the job statement as appropriate.
2. If you are using a different compiler or Language Environment, you need to replace the CBC and CEE data set names with your own. If you use your own JCL, be sure and define the MVS and IBMHOST preprocessor macros either in your application with #define statements or with the DEFINE compile option.
3. Change the TBXFNTDD and FNTLIBDD statements to point to the data sets where your fonts were installed by the system programmer.
4. Modify the application source “outputFilename” to specify a valid output data set and run this job. It produces an AFP document data set “userid.ATXCPPTS.AFP” by default. To print, you must make the page segments and overlays used in the sample available to PSF, either by copying them to the system AFP resource data sets or by specifying a a USERLIB to PSF.

To compile the C++ language sample program:

```

//xxxCPTST JOB (TSS40000),'xxx',MSGLEVEL=(1,1),NOTIFY=xxx,CLASS=A,
//          MSGCLASS=T
//*
//** SAMPLE JCL TO COMPILE, PRELINK, LINK & RUN
//** THE SAMPLE C++ PROGRAM ATXCPTST, THE
//** WOMBAT SAVINGS AND LOAN EXAMPLE
//**
//-----
//*   COMPILE STEP:
//-----
//COMPILE EXEC PGM=CBCCRVR,REGION=0K,
//      PARM=(' /CXX OPTFILE(DD:CCOPT) ')
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
//          DD DSN=CBBC.SCBCCMP,DISP=SHR
//SYSMSGSD DSN=CBBC.SCBCCMSG(EDCMSG),DISP=SHR
//SYSXMSGSD DSN=CBBC.SCBCCMSG(CBCCMSG),DISP=SHR
//SYSIN DD DSN=ATX.SATXSAMP(ATXCPTST),DISP=SHR
//SYSLIN DD DSN=ATX.LOADSET,UNIT=VIO,
//          DISP=(MOD,PASS),SPACE=(512,(50,20)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSCPRT DD SYSOUT=*
//SYSUT1 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT6 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10 DD SYSOUT=*
//SYSUT14 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT15 DD SYSOUT=*
//** COMPILER OPTIONS
//CCOPT DD *
//          SYSPATH(/CEE/SCEEH,/CBC/SCLBH,/ATX/SATXCHDR/DOCUMENT)
//          USERPATH(/ATX/SATXCHDR/DOCUMENT)
//          DEFINE(IBMHOST)

```

```

DEFINE(MVS)
OPT
/*
/*-----
/** PRE-LINKEDIT STEP:
/*-----
//PLKED EXEC PGM=EDCPLRK,REGION=2048K,COND=(8,LT,COMPILE),
// PARM='MAP,NOER,NONCAL'
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
//SYSMSGSGS DD DSN=CEE.SCEEMSGP(EDCPMSGGE),DISP=SHR
//SYSLIB DD DSN=CEE.SCEECPP,DISP=SHR
//SYSIN DD DSN=*.COMPILE.SYSLIN,DISP=(OLD,DELETE)
// DD DSN=CBC.SCLB3SID(IOSTREAM),DISP=SHR
// DD DSN=CBC.SCLB3SID(APPSUPP),DISP=SHR
// DD DSN=CBC.SCLB3SID(COLLECT),DISP=SHR
// DD DDNAME=SYSIN2
//MYDLLS DD DSN=ATX.SATXIMP,DISP=SHR
//SYSMOD DD DSN=&&PLKSET,UNIT=VIO,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD DUMMY
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN2 DD *
INCLUDE MYDLLS(ATXTBOX)
NAME ATXCPTST(R)
/*
/*-----
/** LINKEDIT STEP:
/*-----
//LKED EXEC PGM=HEWL,REGION=1024K,COND=(8,LE,PLKED),
// PARM='AMODE=31,RMODE=ANY,MAP,RENT'
//SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR
//SYSLIN DD DSN=*.PLKED.SYSMOD,DISP=(OLD,DELETE)
//SYSLMOD DD DSN=*&&GOSET(GO),SPACE=(TRK,(10,10,1)),
// UNIT=SYSDA,DISP=(NEW,PASS)
//SYSUT1 DD UNIT=VIO,SPACE=(32000,(30,30))
//SYSPRINT DD SYSOUT=*
/*
/*-----
/** GO STEP:
/*-----
//GO EXEC PGM=*.LKED.SYSLMOD,
// REGION=0K,
// COND=((8,LT,COMPILE),(4,LT,LKED))
//STEPLIB DD DSN=ATX.SATXLOAD,DISP=SHR
// DD DSN=CEE.SCEERUN,DISP=SHR
// DD DSN=CBC.SCLB3DLL,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//FNTLIBDD DD DSN=SYS1.FONTLIBB,DISP=SHR
//TBXFNTDD DD DSN=xxx.TBOX.FONTS,DISP=SHR
//SAMPDATA DD DSN=ATX.SATXDATA(SMTNTPAT),DISP=SHR

```

The following is an example of CBLGetBuffer that uses the buffered interface to return AFP structured fields to your program:

382 AFP Toolbox User's Guide

```

*****
PROGRAM-ID. "ATXGETB".
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM.
OBJECT-COMPUTER. IBM.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OUTPUT-DATA
        ASSIGN TO OUTFILE
        ORGANIZATION IS SEQUENTIAL
        FILE STATUS IS OUT-FILE-STATUS.

*****
*
*                               DATA DIVISION
*
*****
DATA DIVISION.
FILE SECTION.
    FD OUTPUT-DATA
        BLOCK CONTAINS 0 RECORDS
        RECORDING MODE V
        RECORD IS VARYING DEPENDING ON AFPOUT-REC-LEN
        LABEL RECORDS STANDARD.
    01  OUTPUT-RECORD.
    03  FILLER                                OCCURS 9 TO 8201 TIMES,
                                                DEPENDING ON AFPOUT-REC-LEN,
                                                PIC X.

WORKING-STORAGE SECTION.

    01  OUT-FILE-STATUS                        PIC 99.
    01  AFPOUT-REC-LEN                        PIC 9999 BINARY.
    01  FONT-IDS.
*   12pt Times New Roman defined by attributes
    05  TIM12TYPE                            PIC 9(4) BINARY VALUE 0.

COPY ATXCBBVAR.

/-----*
*****
*
*   MAINLINE
*
*****
PROCEDURE DIVISION.

MAINLINE.
    DISPLAY "ATXGETB " UPON SYSOUT.

    OPEN OUTPUT OUTPUT-DATA.
    IF OUT-FILE-STATUS NOT = ZEROS
        DISPLAY "UNABLE TO OPEN OUTPUT FILE"
        DISPLAY "FILE STATUS=" OUT-FILE-STATUS
        STOP RUN.

    PERFORM AFPINIT.

*-----*
*   Do BgnDoc
*-----*
    MOVE "COBOL TEST DOC" TO AFP-DOC-NAME.
    MOVE "This is a comment" TO AFP-DOC-COMMENT.

```

## CBLGetBuffer Program

```
        MOVE BUFFERED TO AFP-OUTPUT-TYPE.
        PERFORM AFPBDOC.
        MOVE "T1V10500" TO AFP-CODE-PAGE.
        MOVE "TIMES NEW ROMAN LATIN1" TO AFP-DESCRIPTIVE-NAME.
        MOVE 120 TO AFP-POINT-SIZE.
        MOVE MEDIUM TO AFP-WEIGHT.
        MOVE NORML TO AFP-FONT-WIDTH.
        MOVE ROMAN TO AFP-STYLE.
        PERFORM AFPDFNTAT.
        MOVE AFP-FONT-ID TO TIM12TYPE.

        PERFORM WRITE-A-PAGE 10 TIMES.

* EndDoc
        PERFORM AFPEDOC.
* Need to flush that last pesky record from the buffer
        SET AFP-CURRENT-HANDLE TO AFP-DOCUMENT-HANDLE.
        MOVE TRU TO AFP-MORE-RECORDS.
        PERFORM GET-A-BUFFER
        UNTIL AFP-MORE-RECORDS EQUAL FALS.

* EndSession
        PERFORM AFPEND.

        CLOSE OUTPUT-DATA.
        DISPLAY "ATXGETB COMPLETED".
        STOP RUN.

*-----
* PERFORMED PARAGRAPHS
*-----

        WRITE-A-PAGE.
* BgnPage
        PERFORM AFPBPAGE.
        SET AFP-CURRENT-HANDLE TO AFP-PAGE-HANDLE.
* SetPos
        MOVE 1440 TO AFP-X-COORDINATE.
        MOVE 1440 TO AFP-Y-COORDINATE.
        MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.
        MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.
        PERFORM AFPSPPOS.
* SetFont
        MOVE TIM12TYPE TO AFP-FONT-ID.
        PERFORM AFPSFONT.
* Write the 'Hello World' string 25 times
        PERFORM WRITE-THE-STRING 25 TIMES.
* EndPage
        PERFORM AFPEPAGE.
* Now fetch the AFPDS from the buffer and write it to the
* output file
* set MORE-RECORDS flag to TRUE because it is initialized
* to FALSE in ATXCBVAR
        MOVE TRU TO AFP-MORE-RECORDS.
        PERFORM GET-A-BUFFER
        UNTIL AFP-MORE-RECORDS EQUAL FALS.

        WRITE-THE-STRING.
* AFPWriteString
        MOVE "Hello World" TO AFP-CHARACTER-STRING.
        MOVE 11 TO AFP-STRING-LENGTH.
        MOVE LEFT-ALIGN TO AFP-ALIGNMENT-OPTION.
        MOVE TRU TO AFP-POSITION-OPTION.
        MOVE -1 TO AFP-EXTRA-VARSPACE.
        MOVE -1 TO AFP-EXTRA-ICSPACE.
        MOVE "." TO AFP-ALIGNMENT-CHAR.
        PERFORM AFPWRITE.
```



```

        PERFORM AFPNLINE.
* Get the AFPDS out of the buffer
GET-A-BUFFER.
    MOVE 8205 TO AFP-BUFFER-LENGTH.
    PERFORM AFPGBUF.
    MOVE AFP-BUFFER-LENGTH TO AFPOUT-REC-LEN.
*   DISPLAY "BUFFER IS " AFP-BUFFER(1:50).
    IF AFP-MORE-RECORDS = TRU
        WRITE OUTPUT-RECORD FROM AFP-BUFFER.
*-----END OF PERFORMED PARAGRAPHS
    COPY ATXCBPRF.
    COPY ATXSTRL.
    COPY ATXTRIM.

END PROGRAM "ATXGETB".

```

---

## Running the MVS Toolbox on an Open Edition System

You can run the AFP Toolbox as a Unix application under Unix System Services (USS), but this is not a formally supported environment when running on OS/390. Follow these steps to run the AFP Toolbox as a Unix application under USS:

1. Compile and link your C test case. The following example assumes you have copied the sample, ATXCTEST, to your local directory for user ID xxx:

```

cc /var/u/xxx/atxctest.c -o atxctest.o \
-V
-W c,"dll,langlvl(extended)" \
-D MVS -D MVS0E -D IBMHOST \
-I "'ATX.SATXCHDR.DOCUMENT.AFPDOC.AFPTK.H'" \
-I "'ATX.SATXCHDR.DOCUMENT.AFPDOC.H'" \
-I "'ATX.SATXCHDR.DOCUMENT.AFPDOC.OCAS.H'" \
-I "'ATX.SATXCHDR.DOCUMENT.H'" \
-L "'ATX.SATXIMP'"

```

2. Preallocate the necessary DD statements to access the font libraries, sample data, and output file:

```

/* rexx */
call bpxwdyn("ALLOC DD(TBXFNTDD) DA(SYS1.TBOX.FONTS) SHR REUSE")
call bpxwdyn("ALLOC DD(FNTLIBDD) DA(FONTS.FONTLIBB) SHR REUSE")
call bpxwdyn("ALLOC DD(SAMPDATA) DA(ATX.SATXDATA(STMNTDAT)) SHR")
call bpxwdyn("ALLOC DD(CTESTOUT) DA(xxx.TBOX.OUTPUT(USSCTEST)) SHR")

```

3. Set a USS environment variable by issuing:

```

_BPX_SHAREAS=MUST
export _BPX_SHAREAS

```

4. Set up the STEPLIB statement to access the C, C++, and Toolbox load modules:

```

STEPLIB=CEE.SCEERUN:CBC.SCBCCMP:ATX.SATXLOAD

```

5. Execute your program from the Unix command line. This will produce output in the file pointed to by DD CTESTOUT:

```

atxctest

```

## OS/390 Unix System Services Environment Variables

This is the Unix System Services (USS) environment variable set for USS programs that use Toolbox:

```

_CXX_LIBDIRS=/lib /usr/lib
MAIL=/usr/mail/USERID
_BPX_TERMPATH=OMVS
PATH=/usr/local/tools/bin:/usr/lpp/Printsrv/bin:/bin:.
_CXX_WORK_UNIT=SYSDA
_CXX_INCDIRS=/usr/include
SHELL=/bin/sh
_CC_LIBDIRS=/lib /usr/lib
_C89_SLIB_PREFIX=SYS1
_CXX_PLIB_PREFIX=CEE

```

## CBLGetBuffer Program

```
_CC_WORK_UNIT=SYSDA
_CC_INCDIRS=/usr/include
COLUMNS=80
_CC_PLIB_PREFIX=CEE
_=/bin/env
LOGNAME=USERID
STEPLIB=CEE.SCEERUN:CBC.SCBCCMP:ATX.SATXLOAD
_CC_PVERSION=0x21020000
LANG=C
LIBPATH=/usr/local/tools/lib:/usr/lpp/Printsrv/lib:/lib:/usr/lib:.
_CXX_SLIB_PREFIX=SYS1
_C89_LIBDIRS=/lib /usr/lib
TERM=dumb
_BPX_SHAREAS=MUST
_C89_WORK_UNIT=SYSDA
_C89_INCDIRS=/usr/include
HOME=/u/userid
_CC_SLIB_PREFIX=SYS1
LINES=20
_C89_PLIB_PREFIX=CEE
_CXX_PVERSION=0x21020000
JAVA_HOME=/usr/lpp/java/J1.1
TZ=MST7MDT
MANPATH=/usr/local/tools/man:/usr/man/%L:/usr/lpp/Printsrv/man/%L
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lpp/Printsrv/%L/%N
_C89_PVERSION=0x21020000
```

---

## Sending Output Directly to MVS SPOOL

You can direct output from a Toolbox application directly to the MVS spool by using the following DD statement in your JCL (assuming your application is writing to CTESTOUT).

```
//CTESTOUT DD SYSOUT=U,OUTPUT=(*.FORMAT),COPIES=1,
// DCB=(RECFM=VBM,LRECL=8205,BLKSIZE=8209)
```

## Troubleshooting

The most common problems in using the AFP Toolbox are installation related and can be avoided by following the instructions in this chapter and in the program directory. However, if you are unable to install or compile and run the sample programs, or if your program is not working as you expect, here are a few troubleshooting tips to try before calling for service:

1. I get error codes from my AFPDefineFont function calls that say the index file cannot be found, or that either code page files or metrics files are missing.  
The fonts must be installed, the FLIP utility must have been run successfully, and you must point to the data sets with the FNTLIBDD and TBXFNTDD statements when your job is executed.
2. My document prints, but there are no spaces between the words or the line spacing is incorrect.  
The AFP Toolbox is finding the index file (TBXFNTDD) but not the font metrics files (FNTLIBDD). Make sure the fonts are installed and that you have authority to read the data sets.
3. The output from my C++ program won't print.  
The AFP Toolbox uses an ostream to write output from C++ applications. Unfortunately the MVS ostream class does not allow record level I/O, so the AFP records are written with structured fields that cross record boundaries. PSF/MVS cannot print this type of data. You have four options:
  - a. Reblock the output file using the AFRREBLK utility supplied with PSF/MVS
  - b. Download the output file to a workstation for printing with PSF/AIX or PSF/2 (Warp Server)
  - c. Use the buffered interface for AFP output from the Toolbox and write the structured fields yourself using a file instead of a stream
  - d. Use the C functions instead of the C++ objects to build the data stream. You can call the C functions from your C++ application.
4. My COBOL text strings are not printing properly. I am not using the PERFORM statements.  
The underlying AFP Toolbox functions are written in C++, so they expect character strings to be null-terminated (ended with X'00'). If you use the PERFORM statements, the null character is inserted for you. If you are calling the CBL functions directly, make sure that all names and other character strings are ended with a null by adding LOW-VALUES to the end of the string.
5. My dynamic COBOL calls to the AFP Toolbox in OS/390 do not work.

Some changes were made in the OS/390 Language Environment support that affect the way AFP Toolbox calls can be made dynamically. Typically, in MVS, to make a dynamic call the following code is used:

```
COBOL program "A" calls program "B" dynamically:
77 PROGRAMB PIC X(8) VALUE "B".
CALL PROGRAMB.
```

If COBOL program "B" calls AFP TOOLBOX (PERFORM AFPBDOC), then this type of dynamic call will not work.

The way you must call in an OS/390 environment is different because you now need to call "B" as a DLL program rather than with the conventional dynamic link syntax. If you are experiencing problems trying to make dynamic calls, make the following changes to your Toolbox environment:

- a. Assure that PTF UG22925 is applied.
- b. Modify program "A" to make what appears to be a static call to "B". Based on the above example, change the call from CALL PROGRAMB to CALL "B".
- c. Compile "B" as a DLL by adding the "DLL" and "EXPORTALL" compile options. This will create a sidedeck that will be linked to program "A". The sidedeck is created during the PRELINK step and is written to the DD SYSDEFSD. Refer to the Language Environment Programmer's Guide for more information on the DLL compile option.
- d. Modify the JCL for A to include the sidedeck for B in the prelink step:

```
//SYSIN2 DD *
INCLUDE SDECK (9B)
NAME A(R)
```

## Troubleshooting

- e. Modify the JCL for B to NOT include the ATXCOBOL object in the prelink:

```
//SYSIN2 DD *  
INCLUDE SDECK (ATXTBOX)  
NAME B(R)
```

- 6. No blanks appear in the printed output.

If you are using COBOL processing then make sure that the variable space parameters are initialized correctly.

The following two lines need to be added before the first AFPWRITE command (just before the AFPDFNAM):

```
MOVE DEFAULT-WS TO AFP-EXTRA-VARS-SPACE.  
MOVE DEFAULT-ICS TO AFP-EXTRA-IC-SPACE.
```

- 7. My program is running sluggish or I run out of storage, the program calls Define Font within a loop.  
Call Delete Font to get rid of the font before re-defining it. See Delete Font in Chapter 3, “AFP Toolbox Language Reference” on page 53 for more information.
- 8. I was expecting a '!' or a '!' but I get ' ] '.

The Toolbox assumes by default that all input on MVS is encoded in codepage T1V10500. Codepage T1V10500 has a square bracket at location x'5A'. Unfortunately, old EBCDIC has an explanation point at x'5A', so if you type in a '!' from the keyboard it appears as input x'5A' that is translated to a ']' at print time. The solution is to use another input codepage, for example T1V10037, using the Set Text Codepage function. For best performance, you should also change the output codepage from the Define Font By Attribute call.

---

## Appendix B. AFP Toolbox and the AIX Environment

This appendix contains the following information about using AFP Toolbox in the AIX environment:

- What prerequisites and system requirements you need
- How to define fonts
- How to compile and run sample programs
- How to view online documentation
- Some useful tools you can use
- How to update your font index
- Some troubleshooting information.

**Note:** The sample programs in this publication are presented only as samples, use the members that are actually installed on your system for testing purposes.

---

### The FontMeister Directory

The FontMeister directory is where the FontMeister Font Management shared library is located. The FontMeister program manages the font information installed from the AFP Font Collection CD. We recommend that you keep this shared library in a directory separate from AFP Toolbox. See “Troubleshooting” on page 387 for additional information about FontMeister.

---

### Compiling and Running the Sample Programs

There are sample programs included with AFP Toolbox for the C and C++ languages. You should compile and run these sample programs, both as an exercise to familiarize yourself with AFP Toolbox and as a verification of the installation procedures. See Chapter 2, “Using AFP Toolbox” on page 13 for examples of the output from the sample programs and a description of their function calls.

To compile and run the C language sample program, change directories to the `/usr/lpp/toolbox/samples` directory and issue the following command:

```
make -f ctest1.mak
```

When completed, this makefile will produce an executable called **ctest1**. Run this program. It produces an output file called **ctest1.afp**. Compare this file to the file **ctest1.cmp**. Try to use AFP Workbench for Windows to view the **ctest1.afp** file.

To compile and run the C++ language sample program, issue the following command:

```
make -f cpptest1.mak
```

When completed, this makefile will produce an executable called **cpptest1**. Run this program. It produces an output file called **cpptest1.afp**. Compare this file to the file **cpptest1.cmp**. In order to view the file, copy the WSLLOGO.PSF file to the resource directory of AFP Workbench. If you do not, the logo will not appear and you will get a warning message. In order to print the file, the WSLLOGO.PSG file must be made available to PSF.

---

### Viewing the Online Documentation

As part of your product package, you received a copy of AFP Workbench for OS/2 and Windows for viewing the online documentation provided with AFP Toolbox. AFP Workbench runs under both Windows and Win-OS2 when running OS/2.

Refer to the installation instructions included with AFP Workbench for installing this product on a workstation. After the AFP Workbench is installed, you can use it to view the online documentation included in the `/BOOKS` directory on the IBM AFP Toolbox CD-ROM.

## AFP Toolbox and AIX

The *IBM AFP Toolbox for Multiple Operating Systems User's Guide* is in the file ACEU1MST.AFP. There are also PostScript (ACEU1MST.PS), plain text (ACEU1MST.TXT), and BookManager® (ACEU1MST.BOO) versions available. Refer to the CD-ROM booklet for more information about viewing the online documentation.

Information about using the BookManager Reader to view the online documentation is available in the READER.TXT file in the /BOOKS directory.

---

## Useful Tools

Included with this copy of IBM AFP Toolbox are some useful tools for managing your AFP Toolbox development system and diagnosing problems in the AFP documents you produce. These tools are located in the /aix/tools directory of your IBM AFP Toolbox CD-ROM.

**Note:** These tools are unsupported additions to AFP Toolbox, included for your convenience.

## AFPDMP

The AFPDMP utility is a program for displaying the structured field contents of an AFP file in readable format. It is identical to the AFPDMP program provided with the PSF/2 and PSF/6000 products. The syntax of AFPDMP is as follows:

```
/cdrom/aix/tools> afdmp -?  
-? gives help
```

```
usage: afdmp [-d0x] [-R#[-#]] [-Ssfi] [-Xsfi] filename [filename ...]  
dump AFPDS in readable formats  
-d describe record in detail  
-0 give record offset instead of record number  
-x dump record in hex format  
-R select a specific record or range of records  
-S select an SFI to work with  
-X exclude an SFI from being worked with
```

## FLIPLIST

The FLIPLIST utility is the font library index program list tool. It is used to display the list of fonts that are currently defined on your system. This command has no parameters. It can also be used to verify the installation of the FontMeister component of AFP Toolbox. Verification is simple: if FLIPLIST produces output, AFP Toolbox is installed correctly. The output from the FLIPLIST tool defines the valid input combinations for the Define Font By Attribute and Define Font By Name function calls in AFP Toolbox. Here are a few lines of output from the FLIPLIST tool:

```
/cdrom/aix/tools> fliplist
```

Typeface Name	Weight	Style	Width	Deci-Points	Char Set	AFM File
BOLDFACE LATIN1	Bold	Roman	Medium	120	C08400B0	BFC
BOOKMASTER LATIN1	Medium	Roman	Medium	50	C0B20050	EDFBL
BOOKMASTER LATIN1	Medium	Italic	Medium	50	C0B30050	EDFBLI
BOOKMASTER LATIN1	Bold	Roman	Medium	50	C0B40050	EDFBLB
...						

See “Troubleshooting” on page 393 for additional information about the FLIPUPDT utility.

## Updating Your Font Index

By default, AFP Toolbox is set up to use the fonts provided with the AFP Font Collection appropriate for your environment. The font index file shipped with AFP Toolbox is used to tell what fonts are available for use in your applications. FLIPLIST provides a human-readable listing of these fonts.

- I To update your font index by adding or deleting font definitions, follow these steps:

1. To add a font definition, create AFP character set and code page objects for printing your font on AFP printers. You can use IBM's Type Transformer program (part of IBM AFP Font Collection) to create AFP font members from your Adobe Type 1 font and build the appropriate files.
2. Create a flat file that contains the information for your updates. The first six fields are separated by blanks and the last five by slashes ( / ). The pattern technology flag in the last entry field is optional. This file has 12 input fields (0-11):
 

<b>Field 0</b>	Indicates whether to add (a) this font to or delete (d) it from the font index.
<b>Field 1</b>	The point size as an integer in decipoints (point size times ten). The point size should be 0 for an outline font.
<b>Field 2</b>	The weight of the font as a number in the range 1-9 where 1 is ultra light, 5 is medium, and 9 is ultra bold. The possible weight values (column two) are: <ul style="list-style-type: none"> <li>1 = ULTRA_LIGHT_WT</li> <li>2 = EXTRA_LIGHT_WT</li> <li>3 = LIGHT_WT</li> <li>4 = SEMI_LIGHT_WT</li> <li>5 = MEDIUM_WT</li> <li>6 = SEMI_BOLD_WT</li> <li>7 = BOLD_WT</li> <li>8 = EXTRA_BOLD_WT</li> <li>9 = ULTRA_BOLD_WT</li> </ul>
<b>Field 3</b>	The width of the font as a number in the range 1-9 where 1 is ultra condensed, 5 is medium, and 9 is ultra expanded. The possible width values (column three) are: <ul style="list-style-type: none"> <li>1 = ULTRA_CONDENSED_WD</li> <li>2 = EXTRA_CONDENSED_WD</li> <li>3 = CONDENSED_WD</li> <li>4 = SEMI_CONDENSED_WD</li> <li>5 = MEDIUM_WD</li> <li>6 = SEMI_EXPANDED_WD</li> <li>7 = EXPANDED_WD</li> <li>8 = EXTRA_EXPANDED_WD</li> <li>9 = ULTRA_EXPANDED_WD</li> </ul>
<b>Field 4</b>	The font style as a hex number representing two bytes of information. Currently only the high order bit is defined, so specify either 0x8000 for italic or 0x0000 for roman. The possible style flag encoding values (column four) are: <ul style="list-style-type: none"> <li>0X0000 = NORMAL_FS</li> <li>0X0800 = OVERSTRUCK_FS</li> <li>0X1000 = OUTLINE_FS</li> <li>0X4000 = UNDERSCORE_FS</li> <li>0X8000 = ITALIC_FS</li> </ul>
<b>Field 5</b>	The name of the AFM file (file containing the Adobe Type 1 metrics). You must specify a value for this parameter, even if there is no AFM file.
<b>Field 6</b>	The name of the character set file. You must specify a value for this parameter, even if there is no character set file.
<b>Field 7</b>	The typeface or descriptive name of the font from the character set, to be used on the Define Font call.
<b>Field 8</b>	Font Family, from the Adobe font. If this is not an Adobe font, specify TypefaceName again.
<b>Field 9</b>	FontName, from the Adobe font. If this is not an Adobe font, specify TypefaceName again.
<b>Field 10</b>	FullName, from the Adobe font. If this is not an Adobe font, specify TypefaceName again.
<b>Field 11</b>	Pattern Technology is an optional field and defaults to a SBCS raster font. Supported fonts are SBCS Outline or Outline, Raster, and DBCS Outline.
3. Run the FLIPUPDT program for your environment to build a new font index file for your fonts. A file called AFRUSER.FIN is created or modified. For AIX users, move the file to a directory where the shared library for AFP Toolbox is installed, for example /usr/lpp/fontmstr.

## Updating Your Font Index

4. Run FLIPLIST to produce a listing and ensure your new font is shown as valid for AFP Toolbox.
5. When you run your application, make sure that AFP Toolbox has access to your new font index file.  
See the install information for your system to find out how to do this in your environment.

For example, if you want to define an AFP outline font named **My Font** with a character set named **C0N01234**, that is medium width, medium weight, and italic style, your input to the FLIPUPDT program is:

```
A 0 5 5 0x8000 NONE C0N01234 My Font/My Font/My Font/My Font/Outline
```



## Troubleshooting

The most common problems in using AFP Toolbox are installation related and can be avoided by following the instructions in this chapter. However, if you are unable to install or compile and run the sample programs, here are a few troubleshooting tips to try before calling for service:

1. I am getting a protection exception in my C++ program whenever I call a function that takes an enumerated value (defined constants such as `LEFT_ALIGN`, or `MEDIUM_WT`).

The interface to AFP Toolbox and FontMeister requires that you treat enumerations as integer values. C++ allows enumerations to vary in size according to how many defined values there are and what numeric values they represent. You must use the compiler option that specifies 4-byte integers should be used for enums.

2. I am not getting any of the fonts I set in my document.

If you are getting a default Courier 10 point font in your output documents and this is not what you have selected, ensure that your application is checking the return codes from your `Define Font By Attributes()` or `Define Font By Name()` calls. If the return codes are good, then make sure that the two files: **afrcore.fin** and **afrpri.map** are in the **/usr/lpp/fontmstr** directory (or are symbolically linked from files in that directory to their actual locations) and that you have execute permissions to read these files.

3. All of the character strings in my output are printing without blanks between words.

You probably either did not install the FontMeister files into **/usr/lpp/fontmstr**, do not have the fonts installed in **/usr/lpp/afpfonts** and **/usr/lpp/psfonts**, or you do not have permission to read these files at execution time.

4. I do not understand how to add and update fonts for AFP Toolbox.

To update the fonts available to AFP Toolbox you need access to a utility called FLIPUPDT. You can get this tool from our ftp site at <ftp://ftp.software.ibm.com/printers/products/toolbox/> in the directory for your system. Make sure ptf u442528 has been applied.

The fonts do not require any specific code page, but it must be available to AFP Toolbox in the **/usr/lpp/afpfonts/codepage** directory. Fonts should be transmitted in binary and should use the same naming convention as others in the font directories. You have to place an .AFM file with corresponding font metrics in the **/usr/lpp/psfonts/latin** directory and specify the base name of this AFM file in your FLIPUPDT input file.

### FLIPUPDT instructions

issue `flipupdt -?`. This returns a short explanation and shows the command syntax. You need to make a short input file to describe the changes you want to make to the index file. Remember to keep a copy of the original index file, **/usr.lpp/fontmstr/afrcore.fin** so it can be used to correct or change the index in the future.

To use your modified index file, rename it to **/usr/lpp/fontmstr/afrcore.fin**. This is a copy of the FLIPUPDT sample input control file:

```
| A 120 5 5 0x0000 ATM CSET Typeface/Family Name/Font Name/Full Name
| A 150 6 7 0x8000 ABC DEF Times New Roman/Times Family/Times Font/ Times full Name
```

Each column has a specific meaning as follows:

- Column 1: deci-point size
- Column 2: numeric representation of weight
- Column 3: numeric representation of width
- Column 4: numeric encoding of the style flags
- Column 5: the AFM file name
- Column 6: character set name
- Column 7: Typeface name
- Column 8: family name
- Column 9: font name
- Column 10: Fullname.

## Troubleshooting

- I See “Updating Your Font Index” on page 390 for the valid values for columns 2 through 4.
- 5. I am having problems using outline fonts with AIX 4.1.X.  
The wrong version of the FONT INDEX FILE was shipped when the CD for this release was built. Unfortunately, we are not set up to PTF that file because it should only have to be shipped one time. This file is on the ftp site: <ftp://ftp.software.ibm.com/printers/products/toolbox/> in the AIX directory. If you use AIX 4.1, obtain the new **afrcore.fin** file and put it into your **/usr/lpp/fontmstr** directory in order to use the core set of outline fonts. Ensure you use a BINARY download of this file.

---

## Appendix C. AFP Toolbox and the AS/400 Environment

This appendix contains the following information about using the AFP Toolbox in the AS/400 environment:

- What prerequisites and system requirements you need
- How to define fonts on your system
- How to compile and run sample programs
- How to update your font index
- Some troubleshooting information.

**Note:** Throughout this publication, the term AS/400 applies to both AS/400 and iSeries.

---

### Prerequisites and System Requirements

The IBM AFP Toolbox for AS/400 is provided as a feature of the AFP PrintSuite product. PrintSuite is supported in the following AS/400 releases:

- Version 3 Release 2 for IMPI hardware
- Version 3 Release 7 and above for RISC System/6000® hardware

The following compilers can be used for your applications:

- ILE C
- ILE COBOL
- ILE RPG

AFP Font Collection Version 2 (5648-B45) is required to use AFP Toolbox.

---

### Using Fonts with the AFP Toolbox

Before using the AFP Toolbox, you **must** first have installed the AFP Font Collection, which contains the AFP character set and codepage files. These files are required by the AFP Toolbox to properly handle converting text from codepage to codepage and to measure strings of text for exact placement on a page.

Font libraries installed by the AFP Font Collection are required at run time and must be added to your library list before the application is invoked so that the AFP Toolbox has access to the appropriate font information. You must add the font libraries with “edtlibl” or “addlibl” commands.

**Note:** QFNT01 is required to print the sample programs.

Consult the AFP Font Collection documentation to determine which libraries contain the codepages and character sets that your application uses and add these libraries to your library list.

There is also a file installed as part of the AFP Toolbox that is an index of the fonts provided with AFP Font Collection. This file (QAOCL/QAYTBFCFG) must be available to the Toolbox at run time (in your library list).

---

### Compiling and Running the Sample Programs

There are sample programs included with the AFP Toolbox for the ILE C, COBOL, and RPG languages. By default, these were installed in library QAOCL in file QAYTBSAMPS. At least one of these programs should have been run by your system programmer as part of the Install Verification process. We suggest that you compile and run these sample programs as an exercise to familiarize yourself with the AFP Toolbox product.

## COBOL Sample

To compile, link, and run the COBOL language sample program under ILE COBOL, you must do the following (replace *MYLIB* with the appropriate name of your library where the program can be built):

1. Copy the sample (CBLTEST) from QAOCL/QAYTBSPAMP to your own COBOL source file (*MYLIB/QCBLLESRC*).
2. Copy the include files (AOCLCBLH, AOCLCBLP, APQTRIM, and APQSTRL) from QAOCL/QAYTBDECLS and QAOCL/QAYTBSPAMP to *MYLIB/QCBLLESRC*.
3. Create a physical file named "AFPOUT" in *MYLIB*. This file should have the same characteristics as QAOCL/QAYTBRESCS.<sup>6</sup>
4. Modify the CBLTEST file to replace MYLIB in the output filename with the name in your library.
5. Compile and link the program:

```
CRTCLMOD MODULE(MYLIB/CBLTEST) SRCFILE(MYLIB/QCBLLESRC) +
    LINKLIT(*PRC) OPTION(*NOMONOPRC *SRC *XREF) +
    DBGVIEW(*ALL)
CRTPGM MODULE(MYLIB/CBLTEST) PGM(MYLIB/CBLTEST) +
    BNDSRVPGM(QAOCL/QYTBAFPTBX)
```

6. Run the program:

```
OVRDBF FILE(SUNSEEDATA) TOFILE(QAOCL/QAYTBRESCS) MBR(SUNSEEDATA)
CALL PGM(MYLIB/CBLTEST)
```

This should produce the output file specified in step 3 above. You can compare this file to file QAOCL/QAYTBRESCS (sunseedata). You can also print the output using PSF/400 using PRTAFPDTA, if you make the overlay and page segment members in file QAOCL/QAYTBRESCS available to PSF by adding QAOCL to your library list.

## C Sample

To compile, link, and run the C language sample program under ILE C, you must do the following (replace *MYLIB* with the appropriate name of your library where the program can be built):

1. Copy the sample (CTEST) from QAOCL/QAYTBSPAMP to your own C source file (*MYLIB/QCSRC*).
2. Create a physical file named "AFPOUT" in *MYLIB*. This file should have the same characteristics as QAOCL/QAYTBRESCS.<sup>7</sup>
3. Compile and link the program:

```
CRTCMOD MODULE(MYLIB/CTEST) SRCFILE(MYLIB/QCSRC) TEXT('C TEST') +
    OUTPUT(*PRINT) OPTION(*EXPMAC *SHOWINC *NOLOGMSG) FLAG(30) +
    MSGMT(24) CHECKOUT(*PARM) DBGVIEW(*ALL)

CRTPGM MODULE(MYLIB/CTEST) PGM(MYLIB/CTEST) +
    BNDSRVPGM(QAOCL/QYTBAFPTBX)
```

4. Run the program:

```
CALL PGM(MYLIB/CTEST)
```

This should produce the output file specified in step 2 above. You can compare this file to file QAOCL/QAYTBRESCS(SAMPLEMAST); they should be identical. You can also print the output using PSF/400 using PRTAFPDTA, if you make the overlay and page segment members in file QAOCL/QAYTBRESCS available to PSF.

6. By default the COBOL sample will create the following output file: QSYS.LIB/MYLIB.LIB/AFPOUT.FILE/CBLTESTAFP.MBR Modify the name of the output file (AFP-OUTPUT-FILENAME) in the source program CBLTEST if you want to use a different library, file, or member name.

7. By default the C sample will create the following output file: QSYS.LIB/mylib.LIB/AFPOUT.FILE/CTEST1.MBR Modify the "outputDocFileName" variable in the source program CTEST if you want to use a different library, file, or member name.

## RPG Sample

To compile, link, and run the RPG language sample program under ILE RPG, you must do the following (replace *MYLIB* with the appropriate name of your library where the program can be built):

1. Copy the sample (RPGTEST) from QAOCL/QAYTBSAMPS to your own RPG source file (*MYLIB*/QRPGLESRC).
2. Copy the include files (AOCLRPGH and AOCLRPGS) from QAOCL/QAYTBDECLS to *MYLIB*/QRPGLESRC.

3. Modify the program to change "***MYLIB***" in the following line:

```
/QSYS.LIB/MYLIB/AFPOUT.FILE/RPGTEST.MBR
```

to the name of your own library.

4. Create a physical file named "AFPOUT" with a member called RPGTEST as follows:

```
CRTPF FILE (AFPOUT) RCDLEN(92) MBR(RPGTEST) TEXT
('File to receive sample output from AFP Toolbox')
```

5. Compile and link the program:

```
CRTPRPGMOD MODULE(MYLIB/RPGTEST) SRCFILE(MYLIB/QRPGLESRC) +
DBGVIEW(*ALL)
```

```
CRTPGM MODULE(MYLIB/RPGTEST) PGM(MYLIB/RPGTEST) +
BNDSRVPGM(QAOCL/QYTBAFPTBX)
```

6. Run the program:

```
CALL PGM(MYLIB/RPGTEST)
```

This should produce the output file specified in step 3 above. You can compare this file to file

QAOCL/QAYTBRESCS(RPGTESTAFP); they should be identical. You can also print the output using the PRTAFPDTA command.

## Updating Your Font Index

By default, AFP Toolbox is set up to use the fonts provided with the AFP Font Collection appropriate for your environment. The font index file shipped with AFP Toolbox is used to tell what fonts are available for use in your applications. QAOCL/QYTBFUPDT provides a human-readable listing of these fonts.

To update your font index by adding or deleting font definitions, follow these steps:

1. To add a font definition, create AFP character set and code page objects for printing your font on AFP printers. You can use IBM's Type Transformer program (part of IBM AFP Font Collection) to create AFP font members from your Adobe Type 1 font and build the appropriate files.
2. Create a physical file using SEU that contains the information for your updates. The first six fields are separated by blanks and the last five by slashes ( / ). The pattern technology flag in the last entry field is optional. This file has 12 input fields (0-11):

<b>Field 0</b>	Indicates whether to add (a) this font to or delete (d) it from the font index.
<b>Field 1</b>	The point size as an integer in decipoints (point size times ten). The point size should be 0 for an outline font.
<b>Field 2</b>	The weight of the font as a number in the range 1-9 where 1 is ultra light, 5 is medium, and 9 is ultra bold. The possible weight values (column two) are:
	1 = ULTRA_LIGHT_WT
	2 = EXTRA_LIGHT_WT
	3 = LIGHT_WT
	4 = SEMI_LIGHT_WT
	5 = MEDIUM_WT
	6 = SEMI_BOLD_WT
	7 = BOLD_WT
	8 = EXTRA_BOLD_WT
	9 = ULTRA_BOLD_WT

## Updating Your Font Index

- |                 |   |
|-----------------|---|
| <b>Field 3</b>  | The width of the font as a number in the range 1-9 where 1 is ultra condensed, 5 is medium, and 9 is ultra expanded. The possible width values (column three) are:<br>1 = ULTRA_CONDENSED_WD<br>2 = EXTRA_CONDENSED_WD<br>3 = CONDENSED_WD<br>4 = SEMI_CONDENSED_WD<br>5 = MEDIUM_WD<br>6 = SEMI_EXPANDED_WD<br>7 = EXPANDED_WD<br>8 = EXTRA_EXPANDED_WD<br>9 = ULTRA_EXPANDED_WD |
| <b>Field 4</b>  | The font style as a hex number representing two bytes of information. Currently only the high order bit is defined, so specify either 0x8000 for italic or 0x0000 for roman. The possible style flag encoding values (column four) are:<br>0X0000 = NORMAL_FS<br>0X8000 = ITALIC_FS   |
| <b>Field 5</b>  | The name of the AFM file (file containing the Adobe Type 1 metrics). You must specify a value for this parameter, even if there is no AFM file.   |
| <b>Field 6</b>  | The name of the character set file. You must specify a value for this parameter, even if there is no character set file.  |
| <b>Field 7</b>  | The typeface or descriptive name of the font from the character set, to be used on the Define Font call.  |
| <b>Field 8</b>  | Font Family, from the Adobe font. If this is not an Adobe font, specify TypefaceName again.   |
| <b>Field 9</b>  | FontName, from the Adobe font. If this is not an Adobe font, specify TypefaceName again.  |
| <b>Field 10</b> | FullName, from the Adobe font. If this is not an Adobe font, specify TypefaceName again.  |
| <b>Field 11</b> | Pattern Technology is an optional field and defaults to a SBCS raster font. Supported fonts are SBCS Outline or Outline, Raster, and DBCS Outline.  |
3. Build a new font index file for your fonts:
    - a. Enter call pgm (QAOCL/QYTBFPDT)
    - b. Press F4 (prompt) for a list of the necessary parameters
    - c. Enter name of the font index file (AFRUSERFIN is the default) you want to create or modify using the -f parameter
    - d. Enter the name of the physical file you created in step 2 on page 397 using the -i parameter
    - e. Enter the name of the new or modified font index file (AFRUSERFIN is the default) you want to create using the -0 parameter.
  4. A file called AFRUSERFIN is created or modified.
  5. Run QAOCL/QAYTBFLIST to produce a listing and ensure your new font is shown as valid for AFP Toolbox.
  6. When you run your application, make sure that AFP Toolbox has access to your new font index file. See the install information for your system to find out how to do this in your environment.

For example, if you want to define an AFP outline font named **My Font** with a character set named **C0N01234**, that is medium width, medium weight, and italic style, your input to the QAOCL/QYTBFPDT program is:

```
A 0 5 5 0x8000 NONE C0N01234 My Font/My Font/My Font/My Font/Outline
```

## Troubleshooting

The most common problems in using the AFP Toolbox are installation related and can be avoided completely by following the instructions contained in this chapter and in the program directory. However, if you are unable to install or compile and run the sample programs, here are a few troubleshooting tips to try before calling for service:

1. I get error codes from my Define Font function calls that say the index file cannot be found, or that either codepage files or metrics files are missing.

The AFP Font Collection must have been installed, the font index file must be available in QAOCL/QAYTBFCFG, and your library list must point to the font libraries (such as QFNT01).

2. My document prints, but there are no spaces between the words or the line spacing is off.

The AFP Toolbox is not finding the font metrics files (font character sets). Make sure the fonts are installed and that your library list points to them.

3. I sent the output file to MVS or VM for printing but it will not print.

You need to reblock the data set on the host into individual records. Run the AFRREBLK utility (available from the PSC home page [www.printers.ibm.com](http://www.printers.ibm.com)) to reblock your data.

**Note:** Be sure to upload the data as a FIXED length file with FORMAT(\*F) on SNDNETF.

4. When I run AFRREBLK I get a message about unexpected EOF.

Ignore it. This is caused by extra blanks at the end of the last structured field.

**Note:** Be sure to upload the data as a FIXED length file with FORMAT(\*F) on SNDNETF.

5. My COBOL or RPG text strings are not printing properly.

The underlying AFP Toolbox functions are written in C, so they expect character strings to be null-terminated (ended with a hex X'00' character). If you use the PERFORM statements, the null character is inserted for you. Otherwise, make sure that all names and other character strings are ended with a null by adding LOW-VALUES to the end of the string.

6. How do I add or update fonts to the AFP Toolbox?

To update the fonts available to AFP Toolbox you will need to run a program called QAOCL/QYTBFPDPT.

### Using QAOCL/QYTBFPDPT

Call pgm(qaocl/qytfupdt). This will bring up the command so you can enter parameters and access help. You need to make a short input file to describe the changes you want to make to the index file. Remember to keep a copy of the original font index file, QAYTBFCFG in library QAOCL so it can be used to correct or change the index in the future. You can enter - ? to return a short explanation and show the command syntax.

To use your modified index file, you must rename it to QAYTBFCFG. Here is a copy of the QAOCL/QYTBFPDPT sample input control file. It is 2 lines long:

```
| A 120 5 5 0x0000 ATM CSET Typeface/Family Name/Font Name/Full Name
| A 150 6 7 0x8000 ABC DEF Times New Roman/Times Family/Times Font/ Times full Name
```

Each column has a specific meaning as follows:

```
|   Column 1: deci-point size
|   Column 2: numeric representation of weight
|   Column 3: numeric representation of width
|   Column 4: numeric encoding of the style flags
|   Column 5: the AFM file name
|   Column 6: character set name
|   Column 7: Typeface name
|   Column 8: family name
|   Column 9: font name
|   Column 10: Fullname.
```

## Updating Your Font Index

- I See “Updating Your Font Index” on page 397 for the valid values for columns 2 through 4.



## Appendix D. Migrating Applications from AFP API to AFP Toolbox

The AFP API was shipped with PSF MVS free of charge on releases 2.1 and 2.2. With PSF for OS/390 Version 3.1 and above, the AFP API feature has been removed. The replacement for the API is the AFP Toolbox for MVS.

There is one main function that is in the API and not available in the Toolbox. This is the ability to create areas. The original purpose of areas in the API was to let users create paragraphs of text that could be used more than once within a document. However, most customers used them instead to avoid the API restriction of requiring that text be placed in the page from top-to-bottom order. This restriction does not exist in the Toolbox. There are currently no plans to add the ability to generate areas in the Toolbox, but since more than one page can be built at a time (another ability that was lacking in the API), this requirement has not been raised by customers.

There are also some differences in API calls, so some applications will have to be rewritten. Of course, applications can continue to use the API indefinitely, as long as they have been debugged and are running satisfactorily. New applications should use Toolbox instead.

The following table shows API functions and how they map to similar function calls in the Toolbox. Only parameters that are different between the API and Toolbox are discussed in this table, so if a parameter is not mentioned, the usage is the same on the Toolbox function as it was on the API function.

**Note:** All API functions specify a return code and severity code parameter. In the Toolbox, there is no severity code. All COBOL functions in the Toolbox will set the AFP-RET-CODE variable and will also set the COBOL RETURN-CODE.

Table 12. API Function / Toolbox Equivalent Function

API Function	Toolbox Equivalent Function
Initialize API (AFPINIT)	Begin Session (AFPINIT)
Terminate API (AFPEND, AFPTERM) the API uses AFPEND for normal end and AFPTERM for abnormal end.	End Session (AFPEND) used for any type of termination.
Set Resources Libraries (AFPSLIB) page segment, object, and font libraries	There is no equivalent Toolbox call. Font libraries are specified in JCL with the FNTLIBDD card. Included page segment and objects can each specify the DD name or a fully qualified DSN on the appropriate Toolbox call.
Set Output Characteristics (AFPSOUT) <ul style="list-style-type: none"><li>output data set name</li><li>output record size</li></ul>	Begin Document (AFPBDOC) <ul style="list-style-type: none"><li>specifies output type and DD name</li><li>record size determined by output file DCB</li></ul>
Begin Document (AFPBDOC) <ul style="list-style-type: none"><li>unit of measurement for input numbers</li><li>doc page width &amp; depth</li><li>page orientation</li></ul>	Begin Document (AFPBDOC) <ul style="list-style-type: none"><li>uom: use Set Units</li><li>width &amp; depth: use Set Media Size</li><li>page orientation: not available</li></ul>
End Document (AFPEDOC)	End Document (AFPEDOC)  same call, but if buffered output is used, in the Toolbox the user must explicitly read each record with AFPGBUF.
Begin Group (AFPBGRP)	Begin Group (AFPBGRP)
End Group (AFPEGRP)	End Group (AFPEGRP)

## Migrating Applications

Table 12. API Function / Toolbox Equivalent Function (continued)

Put Tag (AFPPTAG)	Put Tag (AFPPTAG)
Begin Page (AFBPBAG)	Begin Page (AFBPBAG)
parameters for width, depth, orientation	use Set Media Size and Set Text Orientation calls
End Page (AFPEPAG)	End Page (AFPEPAG)
Define Font (AFPDFNT)	Define Font By Attributes (AFPDFNT)
all parameters are the same except there is no "rotation" parm in the Toolbox	Toolbox also lets fonts be defined by name, and supports outline fonts, ASCII codepages, and DBCS fonts, which the API does not.
Get Output Buffer (AFPGBUF)	Get Buffer (AFPGBUF)
<ul style="list-style-type: none"> <li>specify buffer address</li> <li>automatically returns EDT on AFPEDOC</li> </ul>	<ul style="list-style-type: none"> <li>buffer address on AFPBDOC</li> <li>must issue AFPGBUF after AFPEDOC until all structured fields are returned</li> </ul>
Invoke Medium Map (AFPINVM)	Invoke Medium Map (AFPINVM)
ignores call if same as previous IMM	does not ignore multiple calls
Query Attributes (AFPQATT) returns:	Query (AFPQUERY) returns:
units of measure, current x,y position, color, rule thickness, font id, character, and word spacing	current x,y position, color, font id, font linespacing
Query Position (AFPQPOS) returns:	Query (AFPQUERY) returns:
current x,y position	current x, y position (among other things)
Query String Size (AFPQSTR)	Measure String (AFPMSTR)
	Toolbox also has additional function Translate and Measure String (AFPXMSTR) which will translate from one codepage to another and measure. The toolbox also supports DBCS strings.
Begin Paragraph (AFBPBAR)	Begin Paragraph (AFBPBAR)
<ul style="list-style-type: none"> <li>paragraph handle</li> <li>first line indent, offset, margin</li> <li>rule offsets, framing</li> <li>shading</li> <li>format options, line spacing, width, depth</li> </ul>	<ul style="list-style-type: none"> <li>handle not needed</li> <li>not supported</li> <li>use AFPBBOX, AFPEBOX</li> <li>use AFPBSHAD, AFPESHAD</li> <li>equivalent parameters</li> </ul>
Put Text (AFPPTXT)	Put Text (AFPPTXT)
<ul style="list-style-type: none"> <li>concatenate and underline</li> <li>string and remaining string</li> </ul>	<ul style="list-style-type: none"> <li>not supported</li> <li>equivalent parameters, length not required</li> </ul> <p>Toolbox also supports DBCS strings</p>
End Paragraph (AFPEPAR)	End Paragraph (AFPEPAR)
paragraph depth	not returned, use AFPQUERY
Put Box (AFPPBOX)	Put Box (AFPPBOX)
Shading pattern and intensity	not supported, use AFPPutShade
	Toolbox also provides Bgn/End Box to create variable size boxes
Put Character String (AFPPCHS)	Write String (AFPWRITE)
	Toolbox also supports DBCS strings. Underscoring is done with Begin Underscore (AFPBUSC) and End Underscore (AFPEUSC) functions.

Table 12. API Function / Toolbox Equivalent Function (continued)

Put Rule (AFPPRUL)	Put Horizontal Rule (AFPHRUL), Put Vertical Rule (AFPVRUL) Toolbox also supports Bgn/End Rule for drawing variable length rules
Include Object (AFPIOBJ) only GOCA and IOCA objects allowed	Include Object (AFPIOBJ) also allows BCOCA, EPS, TIFF, and so on
Include Overlay (AFPIOVL)	Include Overlay (AFPIOVL) also allows rotated overlays (AFPROVL)
Include Page Segment (AFPIPSG) inline option	Include Page Segment (AFPIPSG) use Image Inline (AFPIIMG) function
Set Color (AFPSCLR) OCA colors only	Set Color (AFPSCLR) Toolbox also supports process color with AFPBCLR/ECLR and AFPPCLR calls
Set Font (AFPSFNT)	Set Font (AFPSFNT)
Set Intercharacter Spacing (AFPSICS)	no equivalent
Set Position (AFPSPOS)	Set Position (AFPSPOS) Toolbox also provides functions for moving independently horizontally & vertically and using current font line spacing (Next Line)
Set Rule Thickness (AFPSRTH)	no specific call but rule thickness can be specified on all calls that generate rules
Set Units (AFPSUNI)	COBOL Set Units (AFPSUNI)
Set Word Spacing (AFPSWSP)	no equivalent
Define Field (AFPDFLD) • shading pattern • field rotation 0, 90, 180, 270	Define Field (AFPDFLD) • not supported • non-zero rotation is a known requirement.
Define Row (AFPDROW) arrangement array specified as two-dimensional	Define Row (AFPDROW) arrangement array specified as one-dimensional, but equivalent function
Begin Table (AFPBTL) table rotation 0, 90, 180, 270	Begin Table (AFPBTL) non-zero rotation is a known requirement.
End Table (AFPETBL)	End Table (AFPETBL)
Begin Row (AFPBROW)	Begin Row (AFPBROW)
End Row (AFPEROW)	End Row (AFPEROW)
Begin Field (AFPBFLD)	Begin Field (AFPBFLD)
End Field (AFPEFLD)	End Field (AFPEFLD)

## Differences in Codepage Processing

The API assumes that all input text is in “green card” EBCDIC code page. By default, Toolbox assumes that input from MVS and AS/400 is in codepage T1V10500 and input from workstation systems is in T1000850. There was no way to change the input codepage for the API, but Toolbox has the Set Input Codepage and Set Text Codepage functions that let you control what codepage is assumed for input strings.

## Migrating Applications

Toolbox is much more flexible, but there is a problem that you might find when migrating API to Toolbox applications. There are several characters that are mapped differently from “green card” to codepage 500. The most common character is the exclamation point ( ! ), which is at codepoint X'5A' on the EBCDIC system but is at X'4B' in T1V10500. When you encounter this problem, an exclamation point entered from the keyboard is printed as a square bracket ( ] ) because X'5A' in T1V10500 is a bracket.

The solution to this problem is to specify a different input codepage, such as T1V10037, by using the Set Text Codepage function call. With that codepage the exclamation point is mapped as it was in API. For best performance, use the same codepage for output strings (Define Font command). Otherwise, Toolbox spends time translating from one codepage to another.

---

## Functions Provided in Toolbox That Are Not in API

1. Generate bar codes (BCOCA) objects
2. Generate graphic (GOCA) objects
3. Draw variable size boxes and rules
4. Create text and rectangles using process color (RGB, CMYK, CIELab, Highlight color)
5. Create shaded blocks using percentage shading for device independence
6. Convert TIFF files to IOCA for printing
7. Process and format DBCS (double byte) strings
8. Handle ASCII codepages
9. Allow user control of input codepages and translate character strings from input to output codepage
10. Define fonts by name (codepage and character set) as well as attributes
11. Define and process outline fonts and scale text to non-integer point sizes
12. Generate output comments (NOP structured fields)
13. Rotate page overlays
14. Repeat strings of text to a specified width (for example, print leader dots from here ..... to here)
15. Generate multiple pages and documents simultaneously
16. Create output in process color, for example, for the Infoprint Color 100 Plus printer.
17. Delete unneeded pages from a document
18. Delete font definitions when no longer needed
19. Set Text Orientation for text within the page
20. Begin and end underscoring
21. Include non-AFP resources for printing

**Note:** The AFP Toolbox will continue to be enhanced to support changes to the MO:DCA architecture. The AFP API has been functionally stabilized and no new enhancements will be made.

---

## Appendix E. AFP Toolbox Table Samples

This appendix contains the complete code to create a table using AFP Toolbox. There are three samples:

- “Sample in C”
- “Sample in C++” on page 450
- “Sample in COBOL” on page 462

---

### Sample in C

```
/******  
*  
* IBM AFP Toolbox Table Processing  
*  
* Sample Header File TBLSAMP1.H  
*  
******/  
  
/*****/  
/* TABLE HEADER FIELDS */  
/*****/  
unsigned int TableIsDefined;  
unsigned int NbrPages;  
unsigned int PageNbr;  
unsigned int ProcessedData1;  
unsigned int ProcessedData2;  
unsigned int TableProcessError;  
  
short  HdrFld1, HdrFld2, HdrFld3;  
short* HdrFldArray[3];  
int HdrFldCnt = 3;  
  
/*****/  
/* TABLE TRANSACTION FIELDS */  
/*****/  
short  TrnFld1, TrnFld2, TrnFld3;  
short* TrnFldArray[3];  
int TrnFldCnt = 3;  
  
float fTransFld = 0.00;  
float fTransTotal = 0.00;  
  
/*****/  
/* TABLE ROW ID'S */  
/*****/  
short HdrRowId;  
  
#define NBRTBLTRANSROWS 45  
short TrnsRows[NBRTBLTRANSROWS];  
  
short SumRowId;  
  
/*****/  
/* TABLE SUMMARY FIELDS */  
/*****/  
short  SumFld1, SumFld2;  
short* SumFldArray[2];  
int SumFldCnt = 2;  
  
/* The RowArrange # of array elements must reflect */  
/* the max # of previously defined subwows and columns. */  
unsigned int RowArrange[3];  
  
/* The MinSubrowDepths # of array elements must reflect */
```

## C Sample

```
/* the NbrofSubrows count. Ex., if NbrofSubrows = 1, */
/* define MinSubrowDepths[] as MinSubrowDepths[1]. */
unsigned int MinSubrowDepths[1];
unsigned int ColumnWidths[3];

unsigned int NbrofColumns;
unsigned int NbrofSubrows;
unsigned short CurrTableDepth;

/*****
/* TABLE PARAMETERS FOR FIELDS/ROWS */
*****/
long TopThick, BotThick, LeftThick, RightThick;
long LeftMargin, RightMargin;
short ShadeIntense;
SHADE ShadePattern;

AlignmentOption FormatOption;
VertAlignOption VertFormat;
TextOrientation TxtOrient;

long AlignPos, LineSpace;

int RetCode;
int Index;
char szTextstring[75];

TBHANDLE hTbl;
TBHANDLE hPage;
TBHANDLE hOutputDoc;
/*****
*
* IBM AFP Toolbox Table Processing
*
* Sample Header File TBLSUBR1.H
*
*****/

/*****
/* TABLE HEADER FIELDS */
*****/
unsigned int TableIsDefined;
unsigned int NbrPages;
unsigned int PageNbr;
unsigned int ProcessedData1;
unsigned int ProcessedData2;
unsigned int TableProcessError;

short Hdrfld1, Hdrfld2, Hdrfld3;
short Hdrfld4, Hdrfld5, Hdrfld6;
short Hdrfld7, Hdrfld8, Hdrfld9, Hdrfld10;
short* HdrFldArray[10];

unsigned int NbrofHdrSubrows = 3;
unsigned int NbrofHdrColumns = 6;
unsigned int HdrColWidths[6];

/* Use # of hdr Rows 3 & # of hdr Cols 6 for array size[24] */
unsigned int HdrRowArrange[24];
int HdrFldCnt = 10;

/*****
/* TABLE TRANSACTION FIELDS */
*****/
short Trnfld1, Trnfld2, Trnfld3;
short Trnfld4, Trnfld5, Trnfld6;
short* TrnFldArray[6];
```

```

int    TrnFldCnt = 6;

float fTransFld = 0.00;
float fTransTotal = 0.00;

/*****
/* TABLE ROW ID'S */
*****/
short HdrRowId;

/*****
/* MAY NOT NEED 45 BUT DEFINE ANYWAY */
*****/
#define NBRTBLTRANSROWS 45

short TrnsRows[NBRTBLTRANSROWS];

short SumRowId;

/*****
/* TABLE SUMMARY FIELDS */
*****/
short SumFld1, SumFld2;
short* SumFldArray[2];
int    SumFldCnt = 2;

/* The RowArrange # of array elements must reflect */
/* the max # of previously defined subrows and columns. */
unsigned int RowArrange[6];

/* The MinSubrowDepths # of array elements must reflect */
/* the NbrofSubrows count. Ex., if NbrofSubrows = 1, */
/* define MinSubrowDepths[] as MinSubrowDepths[1]. */
unsigned int MinSubrowDepths[1];
unsigned int ColumnWidths[6];
unsigned int NbrofColumns;
unsigned int NbrofSubrows;

unsigned short CurrTableDepth;

/*****
/* TABLE PARAMETERS FOR FIELDS/ROWS */
*****/
long TopThick, BotThick, LeftThick, RightThick;
long LeftMargin, RightMargin;

AlignmentOption FormatOption;
VertAlignOption VertFormat;
TextOrientation TxtOrient;
short ShadeIntense;
SHADE ShadePattern;

long AlignPos, LineSpace;

int RetCode;
int Index;

char szCustName[70];
char szYearOfOrder1[10];
char szYearOfOrder2[10];
char szYearOfOrder3[10];
char szGroupOrderSz1[10];
char szGroupOrderSz2[10];
char szTextstring[75];

TBHANDLE hTbl;

```

## C Sample

```
TBHANDLE hPage;
TBHANDLE hOutputDoc;
/*****
*
* IBM AFP Toolbox Table Processing
*
* Sample Source File TBLSAMP1.C
*
*****/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#ifdef __BORLANDC__
#include <mem.h>
#else
#include <memory.h>
#endif

#include "aocl.h"
#include "tblsamp1.h"

#ifdef AS400
static char szLogicalFileName[] = "/mylib.lib/afpout.file/tblsamp1.mbr";
static char OutputFile[] = "/mylib.lib/afpout.file/buffer.mbr";
char inputCodepage[] = "T1V10500";
#else
static char szLogicalFileName[] = "MYOUTPDD";
static char OutputFile[] = "tblsamp1.afp";
char inputCodepage[] = "T1V10500";
#else
char szLogicalFileName[] = "tblsamp1.afp";
char inputCodepage[] = "T1000850";
#endif

char outputCodepage[] = "T1V10500";
char outputDocComment[] = "AFP Toolbox Bank Statement "
                          "Sample Application Output Document";
byte AfpRecBuffer[8192];
FILE* pFileOut;
OUTTYPE TypeOfOutput = TO_OUTPUT_FILE;

FontID titleFont, addressFont, headingsFont;
unsigned short far *ptitleFont, *paddressFont, *pheadingsFont;

FontID accountNameFont, transactionFont;

#ifdef __BORLANDC__
#define _USERENTRY
#endif

/*****/
/* ERROR INFO */
/*****/
void checkForError(int rc, char* message)
{
    if (rc != 0)
    {
        fprintf(stderr, message);
        fprintf(stderr, "AFP Toolbox return code %d\n\n", rc);
        exit(-1);
    }
}
```



```

/*****
/* DEFINE TABLE HEADER FIELDS */
/*****
int define_table_hdr_flds()
{
    RetCode = 0;

    HdrFldArray[0] = &Hdrfld1
    HdrFldArray[1] = &Hdrfld2
    HdrFldArray[2] = &Hdrfld3

    //FormatOption = CENTER_ALIGN;
    FormatOption = LEFT_ALIGN;
    VertFormat    = VERCENTER;
    TxtOrient     = IOB90_T0;
    AlignPos = 0;

    // LeftMargin = 0.0;
    LeftMargin = 0;
    RightMargin = 0;

    LineSpace = 1;

    //ShadeIntense = 0.0;
    ShadeIntense = 18.0;
    ShadePattern = NO_SHADE;

    /* Build Header Fields */
    for(Index = 0; Index < HdrFldCnt; Index++)
    {
        RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
            LeftMargin, RightMargin, LineSpace, TxtOrient, ShadeIntense,
            HdrFldArray[Index]);

        if (RetCode)
        {
            printf("\ndefine_table_hdr_flds Error1 !\n");
            printf("\nError return code: %d", RetCode);
            return(1);
        }
    }

    return(0);
}

/*****
/* DEFINE TABLE HEADER ROW */
/*****
int define_table_hdr_row()
{
    NbrofSubrows = 1;
    NbrofColumns = 3;

    /* Load global values */
    TopThick    = MM_2_U1440(.5);
    BotThick    = MM_2_U1440(.5);
    LeftThick   = MM_2_U1440(.5);
    RightThick  = MM_2_U1440(.5);

    ColumnWidths[0] = MM_2_U1440(40.0);
    ColumnWidths[1] = MM_2_U1440(80.0);
    ColumnWidths[2] = MM_2_U1440(40.0);

    MinSubrowDepths[0] = 1.0; /* MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH ? */

    RowArrange[0] = Hdrfld1; /* MOVE FIELD TO AFP-ROW-ARRANGE */
    RowArrange[1] = Hdrfld2;

```

## C Sample

```
RowArrange[2] = HdrFld3;

RetCode= AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick,
    BotThick, NbrofColumns, NbrofSubrows, RowArrange,
    ColumnWidths, &HdrRowId);

if (RetCode)
{
    printf("\ndefine_table_hdr_row Error !\n");
    printf("\nError return code: %d", RetCode);
    return(1);
}

return(0);
}

/*****
/* DEFINE TABLE TRANSACTION FIELDS */
*****/
int define_table_trans_flds()
{
    TrnFldArray[0] = &TrnFld1
    TrnFldArray[1] = &TrnFld2
    TrnFldArray[2] = &TrnFld3

    //FormatOption = CENTER_ALIGN;
    FormatOption = LEFT_ALIGN;
    //FormatOption = JUSTIFY_ALIGN;
    VertFormat    = VERCENTER;
    TxtOrient     = IOB90_T0;

    AlignPos = 0;
    ShadeIntense = 0.0;

    LeftMargin  = 1.0;
    RightMargin = 1.0;

    /* Build Transaction Fields */
    for(Index = 0; Index < TrnFldCnt; Index++)
    {
        /* Check for field 3 to change margin and alignment position. */
        if (Index == 2)
        {
            LeftMargin  = 0.0;
            AlignPos = 20;
        }

        RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
            LeftMargin, RightMargin, LineSpace, TxtOrient, ShadeIntense,
            TrnFldArray[Index]);

        if (RetCode)
        {
            printf("\ndefine_table_trans_flds Error !\n");
            printf("\nError return code: %d", RetCode);
            return(1);
        }
    }

    return(0);
}

/*****
/* DEFINE TABLE TRANSACTION ROWS */
*****/
int define_table_trans_rows()
{
```

```

int iRowCnt;

NbrofSubrows = 1;
NbrofColumns = 3;

ColumnWidths[0] = MM_2_U1440(40.0);
ColumnWidths[1] = MM_2_U1440(80.0);
ColumnWidths[2] = MM_2_U1440(40.0);

TopThick = MM_2_U1440(.02);
BotThick = MM_2_U1440(.02);

MinSubrowDepths[0] = 1.0; /* MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH ? */

RowArrange[0] = Trnfl1; /* MOVE FIELD TO AFP-ROW-ARRANGE */
RowArrange[1] = Trnfl2;
RowArrange[2] = Trnfl3;

for (iRowCnt = 0; iRowCnt < NBRTBLTRANSROWS; iRowCnt++)
{
    RetCode= AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick,
        BotThick, NbrofColumns, NbrofSubrows, RowArrange,
        ColumnWidths, &TrnsRows[iRowCnt]);

    if (RetCode)
    {
        printf("\ndefine_table_trans_rows Error !\n");
        printf("\nError return code: %d", RetCode);
        return(1);
    }
}

return(0);
}

/*****
/* DEFINE TABLE SUMMARY FIELDS */
*****/
int define_table_sum_flds()
{
    SumFldArray[0] = &Sumfld1
    SumFldArray[1] = &Sumfld2

    FormatOption = CENTER_ALIGN;
    VertFormat   = VERCENTER;
    TxtOrient    = IOB90_T0;

    AlignPos = 0;
    ShadeIntense = 18.0;

    LeftMargin  = 1.0;
    RightMargin = 1.0;

    /* Build Transaction Fields */
    for(Index = 0; Index < SumFldCnt; Index++)
    {
        /* Check for field 2 to change Margin and alignment position. */
        if (Index == 1)
        {
            LeftMargin  = 0.0;
            AlignPos = 20;
        }
    }

    RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
        LeftMargin, RightMargin, LineSpace, TxtOrient, ShadeIntense,
        SumFldArray[Index]);

```

## C Sample

```
if (RetCode)
{
    printf("\ndefine_table_sum_flds Error !\n");
    printf("\nError return code: %d", RetCode);
    return(1);
}

return(0);
}

/*****
/* DEFINE TABLE SUMMARY ROW */
*****/
int define_table_sum_row()
{
    NbrofSubrows = 1;
    NbrofColumns = 2;

    ColumnWidths[0] = MM_2_U1440(90.0);
    ColumnWidths[1] = MM_2_U1440(90.0);
    ColumnWidths[2] = 0.0;

    TopThick = MM_2_U1440(.5);
    BotThick = MM_2_U1440(.5);

    MinSubrowDepths[0] = 1.0; /* MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH ? */

    RowArrange[0] = Sumfld1; /* MOVE FIELD TO AFP-ROW-ARRANGE */
    RowArrange[1] = Sumfld2;
    RowArrange[2] = 0x00;

    RetCode= AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick, BotThick,
        NbrofColumns, NbrofSubrows, RowArrange, ColumnWidths, &SumRowId);

    if (RetCode)
    {
        printf("\ndefine_table_sum_row Error !\n");
        printf("\nError return code: %d", RetCode);
        return(1);
    }

    return(0);
}

/*****
/* WRITE TABLE HEADER ROW */
*****/
int write_table_hdr_row()
{
    /* Begin Header Row */
    RetCode = AFPBeginRow(hTbl, HdrRowId);

    if (RetCode)
        return(1);

    /* Begin Header Field 1 */
    RetCode = AFPBeginField(hTbl, Hdrfld1);

    if (RetCode)
        return(1);

    /* Put Text in Header Field 1 */
    memcpy(szTextstring, NULL, sizeof(szTextstring));
    strcpy(szTextstring, "Date");
    /* Set alignment CENTER here. */
}
```

```

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld1,
                                szTextstring, headingsFont);
if (RetCode)
    return(1);

/* End Header Field 1 */
RetCode = AFPEndField(hTbl);

if (RetCode)
    return(1);

/* Begin Header Field 2 */
RetCode = AFPBeginField(hTbl, Hdrfld2);

if (RetCode)
    return(1);

/* Put Text in Header Field 2 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Transaction");
/* Set alignment CENTER here. */

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld2,
                                szTextstring, headingsFont);
if (RetCode)
    return(1);

/* End Header Field 2 */
RetCode = AFPEndField(hTbl);

if (RetCode)
    return(1);

/* Begin Header Field 3 */
RetCode = AFPBeginField(hTbl, Hdrfld3);

if (RetCode)
    return(1);

/* Put Text in Header Field 3 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Amount");
/* Set alignment CENTER here. */

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld3,
                                szTextstring, headingsFont);
if (RetCode)
    return(1);

/* End Header Field 3 */
RetCode = AFPEndField(hTbl);

if (RetCode)
    return(1);

/* End Header Row */
RetCode = AFPEndRow(hTbl, HdrRowId, &CurrTableDepth);

if (RetCode)
    return(1);

return(0);
}

/*****
/* WRITE TABLE TRANSACTION ROW */
*****/

```

## C Sample

```
int write_table_trans_row(unsigned int iRowId, char* szDateText,
                        char* szTransaction, float fAmnt)
{
    /* Begin Transaction Row */
    RetCode = AFPBeginRow(hTbl, iRowId);

    if (RetCode)
        return(RetCode);

    /* Write Transaction Field 1 */
    RetCode = AFPBeginField(hTbl, TrnFld1);

    if (RetCode)
        return(RetCode);

    /* Transaction Date */
    memcpy(szTextstring, NULL, sizeof(szTextstring));
    strcpy(szTextstring, szDateText);

    RetCode = AFPPutTextInTableFld(hTbl, TrnFld1, szTextstring,
                                   headingsFont);
    if (RetCode)
        return(RetCode);

    /* End Transaction Field 1 */
    RetCode = AFPEndField(hTbl);
    if (RetCode)
        return(RetCode);

    /* Begin Transaction Field 2 */
    RetCode = AFPBeginField(hTbl, TrnFld2);
    if (RetCode)
        return(RetCode);

    /* Transaction Description */
    memcpy(szTextstring, NULL, sizeof(szTextstring));
    strcpy(szTextstring, szTransaction);

    RetCode = AFPPutTextInTableFld(hTbl, TrnFld2,
                                   szTextstring, headingsFont);
    if (RetCode)
        return(RetCode);

    /* End Transaction Field 2 */
    RetCode = AFPEndField(hTbl);
    if (RetCode)
        return(RetCode);

    /* Begin Transaction Field 3 */
    RetCode = AFPBeginField(hTbl, TrnFld3);
    if (RetCode)
        return(RetCode);

    /* Transaction amount */
    fTransFld = fAmnt;
    fTransTotal += fTransFld;
    memcpy(szTextstring, NULL, sizeof(szTextstring));
    sprintf(szTextstring, "%s%.2f", "$", fTransFld);

    RetCode = AFPPutTextInTableFld(hTbl, TrnFld3,
                                   szTextstring, headingsFont);
    if (RetCode)
        return(RetCode);

    /* End Transaction Field 3 */
    RetCode = AFPEndField(hTbl);
    if (RetCode)
```

```

    return(RetCode);

RetCode = AFPEndRow(hTbl, iRowId, &CurrTableDepth);
if (RetCode)
    return(RetCode);

return(0);
}

/*****
/* WRITE TABLE SUMMARY ROW */
*****/
int write_table_sum_row()
{
/* Begin Summary Row */
RetCode = AFPBeginRow(hTbl, SumRowId);
if (RetCode)
    return(RetCode);

/* Begin Summary Field 1 */
RetCode = AFPBeginField(hTbl, Sumfld1);
if (RetCode)
    return(RetCode);

/* Summary Amount */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Total Amount");

RetCode = AFPPutTextInTableFld(hTbl, Sumfld1,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Summary Field 1 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Write Summary Field 2 */
RetCode = AFPBeginField(hTbl, Sumfld2);
if (RetCode)
    return(RetCode);

/* Summary Total */
memcpy(szTextstring, NULL, sizeof(szTextstring));
sprintf(szTextstring, "%s%.2f", "$", fTransTotal);

RetCode = AFPPutTextInTableFld(hTbl, Sumfld2,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 2 */
RetCode = AFPEndField(hTbl);

if (RetCode)
    return(RetCode);

/* End Summary Row */
RetCode = AFPEndRow(hTbl, SumRowId, &CurrTableDepth);

if (RetCode)
    return(RetCode);

return(0);
}

```

## C Sample

```
/******  
/* PROCESS ROWS(1ST GROUP OF TRANSACTIONS */  
/******  
int process_transaction_data1()  
{  
    char szDate[75];  
    char szTrans[75];  
    float fValue = 0.0;  
  
    /* Transaction Date */  
    memcpy(szDate, NULL, sizeof(szDate));  
    strcpy(szDate, "July 1, 2000");  
    /* Transaction Description */  
    memcpy(szTrans, NULL, sizeof(szTrans));  
    strcpy(szTrans, "IBM Thinkpad for Brian");  
    fValue = 950.00;  
  
    RetCode = write_table_trans_row(TrnsRows[0], szDate,  
                                    szTrans, fValue);  
    if (RetCode)  
        return(RetCode);  
  
    /* Transaction Date */  
    memcpy(szDate, NULL, sizeof(szDate));  
    strcpy(szDate, "July 2, 2000");  
    /* Transaction Description */  
    memcpy(szTrans, NULL, sizeof(szTrans));  
    strcpy(szTrans, "IBM Mouse for Alex");  
    fValue = 50.00;  
  
    RetCode = write_table_trans_row(TrnsRows[1], szDate,  
                                    szTrans, fValue);  
    if (RetCode)  
        return(RetCode);  
  
    /* Transaction Date */  
    memcpy(szDate, NULL, sizeof(szDate));  
    strcpy(szDate, "July 3, 2000");  
    /* Transaction Description */  
    memcpy(szTrans, NULL, sizeof(szTrans));  
    strcpy(szTrans, "IBM Printer for Kathy");  
    fValue = 375.00;  
  
    RetCode = write_table_trans_row(TrnsRows[2], szDate,  
                                    szTrans, fValue);  
    if (RetCode)  
        return(RetCode);  
  
    /* Transaction Date */  
    memcpy(szDate, NULL, sizeof(szDate));  
    strcpy(szDate, "July 5, 2000");  
    /* Transaction Description */  
    memcpy(szTrans, NULL, sizeof(szTrans));  
    strcpy(szTrans, "IBM Phone for Roger");  
    fValue = 75.50;  
  
    RetCode = write_table_trans_row(TrnsRows[3], szDate,  
                                    szTrans, fValue);  
    if (RetCode)  
        return(RetCode);  
  
    /* Transaction Date */  
    memcpy(szDate, NULL, sizeof(szDate));  
    strcpy(szDate, "July 6, 2000");  
    /* Transaction Description */  
    memcpy(szTrans, NULL, sizeof(szTrans));  
    strcpy(szTrans, "IBM Keyboard for Michael");
```



```

fValue = 75.50;

RetCode = write_table_trans_row(TrnsRows[4], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 7, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Risc/6000 for Jeri");
fValue = 75.50;

RetCode = write_table_trans_row(TrnsRows[5], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 8, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Super Computer for the Boss: Bill");
fValue = 7500.50;

RetCode = write_table_trans_row(TrnsRows[6], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 9, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Keyboard for David");
fValue = 75.50;

RetCode = write_table_trans_row(TrnsRows[7], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 10, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Mouse for Brian");
fValue = 75.50;

RetCode = write_table_trans_row(TrnsRows[8], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 11, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Keyboard for Alex");
fValue = 75.50;

```

## C Sample

```
RetCode = write_table_trans_row(TrnsRows[9], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 12, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM DCF Manual for Roger");
fValue = 15.00;

RetCode = write_table_trans_row(TrnsRows[10], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 13, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM OGL Manual for Kathy");
fValue = 15.00;

RetCode = write_table_trans_row(TrnsRows[11], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 14, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Visual C++ for Michael");
fValue = 35.00;

RetCode = write_table_trans_row(TrnsRows[12], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 15, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM PLX Compiler for Alex");
fValue = 35.00;

RetCode = write_table_trans_row(TrnsRows[13], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 16, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "Borland C++ Compiler for Brian");
fValue = 35.00;

RetCode = write_table_trans_row(TrnsRows[14], szDate,
                                szTrans, fValue);
```

```

if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 17, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "AIX Development tools for AIX");
fValue = 35.00;

RetCode = write_table_trans_row(TrnsRows[15], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 18, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "Unix Standard Templates for Michael");
fValue = 35.00;

RetCode = write_table_trans_row(TrnsRows[16], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 19, 2000"); */

/* Transaction Description
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "OS/390 Compilers for Kathy");
fValue = 35.00;

RetCode = write_table_trans_row(TrnsRows[17], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode); */

/* Set process switch */
ProcessedData1 = 1;

return(0);
}

/*****
/* PROCESS ROWS(2ND GROUP OF TRANSACTIONS */
*****/
int process_transaction_data2()
{
    char  szDate[75];
    char  szTrans[75];
    float fValue = 0.0;

    /* Transaction Date */
    memcpy(szDate, NULL, sizeof(szDate));
    strcpy(szDate, "July 23, 2000");
    /* Transaction Description */
    memcpy(szTrans, NULL, sizeof(szTrans));
    strcpy(szTrans, "IBM WebSphere for Michael");
    fValue = 155.00;

    RetCode = write_table_trans_row(TrnsRows[0], szDate,

```

## C Sample

```
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 24, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Pencils for Alex");
fValue = 2.00;

RetCode = write_table_trans_row(TrnsRows[1], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 25, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Pencils for Kathy");
fValue = 2.00;

RetCode = write_table_trans_row(TrnsRows[2], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 26, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Shirt for Kathy");
fValue = 6.00;

RetCode = write_table_trans_row(TrnsRows[3], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 27, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Clock for Brian");
fValue = 4.00;

RetCode = write_table_trans_row(TrnsRows[4], szDate,
                                szTrans, fValue);
if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "July 28, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Computer Case for Brian");
fValue = 10.00;

RetCode = write_table_trans_row(TrnsRows[5], szDate,
                                szTrans, fValue);
if (RetCode)
```

```

    return(RetCode);

    /* Transaction Date */
    memcpy(szDate, NULL, sizeof(szDate));
    strcpy(szDate, "July 29, 2000");
    /* Transaction Description */
    memcpy(szTrans, NULL, sizeof(szTrans));
    strcpy(szTrans, "IBM Desktop for Roger");
    fValue = 50.00;

    RetCode = write_table_trans_row(TrnsRows[6], szDate,
                                    szTrans, fValue);
    if (RetCode)
        return(RetCode);

    /* Transaction Date */
    memcpy(szDate, NULL, sizeof(szDate));
    strcpy(szDate, "July 30, 2000");
    /* Transaction Description */
    memcpy(szTrans, NULL, sizeof(szTrans));
    strcpy(szTrans, "IBM Desktop for Kathy");
    fValue = 50.00;

    RetCode = write_table_trans_row(TrnsRows[7], szDate,
                                    szTrans, fValue);
    if (RetCode)
        return(RetCode);

    /* Transaction Date */
    memcpy(szDate, NULL, sizeof(szDate));
    strcpy(szDate, "July 31, 2000");
    /* Transaction Description */
    memcpy(szTrans, NULL, sizeof(szTrans));
    strcpy(szTrans, "IBM Airline Ticket for Bill");
    fValue = 150.00;

    RetCode = write_table_trans_row(TrnsRows[8], szDate,
                                    szTrans, fValue);
    if (RetCode)
        return(RetCode);

    /* Transaction Date */
    memcpy(szDate, NULL, sizeof(szDate));
    strcpy(szDate, "August 1, 2000");
    /* Transaction Description */
    memcpy(szTrans, NULL, sizeof(szTrans));
    strcpy(szTrans, "IBM Diskettes for Alex");
    fValue = 5.00;

    RetCode = write_table_trans_row(TrnsRows[9], szDate,
                                    szTrans, fValue);
    if (RetCode)
        return(RetCode);

    /* Transaction Date */
    memcpy(szDate, NULL, sizeof(szDate));
    strcpy(szDate, "August 2, 2000");
    /* Transaction Description */
    memcpy(szTrans, NULL, sizeof(szTrans));
    strcpy(szTrans, "IBM CD's for Kathy");
    fValue = 5.00;

    RetCode = write_table_trans_row(TrnsRows[10], szDate,
                                    szTrans, fValue);
    if (RetCode)
        return(RetCode);

```

## C Sample

```
/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "August 3, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM OS/2 Warp for Roger");
fValue = 55.00;

RetCode = write_table_trans_row(TrnsRows[11], szDate,
                                szTrans, fValue);

if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "August 4, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM NT/OS for Alex");
fValue = 55.00;

RetCode = write_table_trans_row(TrnsRows[12], szDate,
                                szTrans, fValue);

if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "August 5, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Linus/OS for Brian");
fValue = 55.00;

RetCode = write_table_trans_row(TrnsRows[13], szDate,
                                szTrans, fValue);

if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "August 6, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM Unix/OS for Michael");
fValue = 55.00;

RetCode = write_table_trans_row(TrnsRows[14], szDate,
                                szTrans, fValue);

if (RetCode)
    return(RetCode);

/* Transaction Date */
memcpy(szDate, NULL, sizeof(szDate));
strcpy(szDate, "August 7, 2000");
/* Transaction Description */
memcpy(szTrans, NULL, sizeof(szTrans));
strcpy(szTrans, "IBM OS390/Linux/OS for Roger");
fValue = 55.00;

RetCode = write_table_trans_row(TrnsRows[15], szDate,
                                szTrans, fValue);

if (RetCode)
    return(RetCode);

/* Set processed switch */
ProcessedData2 = 1;
```

```

return(0);
}

/*****
/* PROCESS TABLE TRANSACTIONS */
*****/
int process_table_transactions()
{
char szDate[75];
char szTrans[75];
float fValue = 0.0;

AFPSetPos(hPage, MM_2_U1440(20), ABSOLUTE_POS,
          MM_2_U1440(80),ABSOLUTE_POS);

AFPVMoveTo(hPage, IN_2_U1440(3.5));

/* Begin Table */
TopThick   = MM_2_U1440(.5);
BotThick   = MM_2_U1440(.5);
LeftThick  = MM_2_U1440(.5);
RightThick = MM_2_U1440(.5);

RetCode = AFPBeginTable(hPage, MM_2_U1440(180.0), MM_2_U1440(80.0),
                        0, TopThick, BotThick, LeftThick, RightThick, &hTbl);

if (RetCode)
    return(0);

/* Write Table Header Row */
RetCode = write_table_hdr_row();

if (RetCode)
    return(0);

/* Start Writing Table Transaction Rows */
if (!ProcessedData1)
{
    RetCode = process_transaction_data1();

    /* Page/Table Depth Error */
    if (RetCode == 37)
    {
        memcpy(szTextstring, NULL, sizeof(szTextstring));
        sprintf(szTextstring, "%s%d",
                "\n Table Depth Error(Data 1) On Page ", PageNbr);
        printf(szTextstring);
        return(0);
    }
}
else
/* Start Writing Table Transaction Rows */
if (!ProcessedData2)
{
    RetCode = process_transaction_data2();

    /* Page/Table Depth Error */
    if (RetCode == 37)
    {
        memcpy(szTextstring, NULL, sizeof(szTextstring));
        sprintf(szTextstring, "%s%d",
                "\n Table Depth Error(Data 2) On Page ", PageNbr);
        printf(szTextstring);

        return(0);
    }
}

```

## C Sample

```
/* Write Table Summary Row */
RetCode = write_table_sum_row();

/* Page/Table Depth Error */
if (RetCode == 37)
{
    memcpy(szTextstring, NULL, sizeof(szTextstring));
    sprintf(szTextstring, "%s%d",
            "\n Table Depth Error(Summary Row) On Page ", PageNbr);
    printf(szTextstring);
    return(0);
}

/* End Table */
RetCode = AFPEndTable(&hTbl, &CurrTableDepth);

if (RetCode)
    return(0);

return(1);
}

/*****/
/* CREATE NEW TABLE */
/*****/
int create_new_table()
{
    /*****/
    /* Define Header Fields */
    /*****/
    RetCode = define_table_hdr_flds();
    if (RetCode)
        return(0);

    /*****/
    /* Define Header Row */
    /*****/
    RetCode = define_table_hdr_row();
    if (RetCode)
        return(0);

    /*****/
    /* Define Transaction Fields */
    /*****/
    RetCode = define_table_trans_flds();
    if (RetCode)
        return(0);

    /*****/
    /* Define Transaction Row */
    /*****/
    RetCode = define_table_trans_rows();
    if (RetCode)
        return(0);

    /*****/
    /* Define Summary Fields */
    /*****/
    RetCode = define_table_sum_flds();
    if (RetCode)
        return(0);

    /*****/
    /* Define Summary Row */
    /*****/
}
```



```

RetCode = define_table_sum_row();
if (RetCode)
    return(0);

/*****
/* Table Definitions Successful ! */
*****/
TableIsDefined = 1;

return(1);
}

/*****
/* CREATE NEW PAGE */
*****/
int create_page()
{
    int rc;
    long lpos;

    char buffer[15];
    char* datestr;

    /*****
    /* SET GLOBAL PAGE HANDLE */
    *****/
    hPage = NULL;

    /*****
    /* Begin a page. */
    *****/
    rc = AFPBgnPage(hOutputDoc, "Table Page", &hPage);

    checkForError(rc, "Error Creating a Page\n");

    /*****
    /* Include Logo Page Segment in upper left corner. */
    *****/
    AFPHMoveTo(hPage, IN_2_U1440(.5));
    AFPVMoveTo(hPage, IN_2_U1440(.5));
    /*AFPInclPSeg(hPage, "WSLLOGO", TRUE);*/

    /*****
    /* Set Font, move and write company address. */
    *****/
    AFPSetFont(hPage, titleFont);

    AFPVMoveTo(hPage, IN_2_U1440(.75));

    AFPHMoveTo(hPage, IN_2_U1440(3.5));

    AFPWriteString(hPage, "IBM Printing Systems", -1, -1, LEFT_ALIGN, ' ', TRUE);

    AFPNextLine(hPage);

    AFPWriteString(hPage, "6300 Diagonal Hwy.", -1, -1, LEFT_ALIGN, ' ', TRUE);

    AFPNextLine(hPage);

    AFPWriteString(hPage, "Boulder, CO 80301", -1, -1, LEFT_ALIGN, ' ', TRUE);

    AFPSetFont(hPage, headingsFont);

    AFPHMoveTo(hPage, IN_2_U1440(.5));

    AFPVMoveTo(hPage, IN_2_U1440(2.5));

```

## C Sample

```
AFPWriteString(hPage, "Table Processing Test", -1, -1, LEFT_ALIGN,
               ' ', TRUE);

AFPNextLine(hPage);

AFPWriteString(hPage, "Developer: Brian Smith", -1, -1, LEFT_ALIGN,
               ' ', TRUE);

AFPNextLine(hPage);
AFPNextLine(hPage);
AFPNextLine(hPage);
AFPNextLine(hPage);

/*****
/* Move under address, print statement dates. */
*****/
AFPHMoveTo(hPage, IN_2_U1440(3.5));
AFPVMoveTo(hPage, IN_2_U1440(2.5));

AFPWriteString(hPage, "Statement From: ", -1, -1, RIGHT_ALIGN, ' ', TRUE);
AFPWriteString(hPage, "July 1, 2000", -1, -1, LEFT_ALIGN, ' ', TRUE);
AFPNextLine(hPage);

AFPHMoveTo(hPage, IN_2_U1440(3.2));

AFPWriteString(hPage, "To: July 26, 2000", -1, -1, LEFT_ALIGN, ' ', TRUE);

/*****
/* Add Bottom Disclaimer to Document */
*****/
AFPVMoveTo(hPage, IN_2_U1440(10.55));
AFPHMoveTo(hPage, IN_2_U1440(8.5/2.0));

AFPWriteString(hPage, "IBM AFP Toolbox Sample Bank "
                    "Statement Application Output",
               -1, -1, CENTER_ALIGN, ' ', TRUE);

/*****
/* Move current position (under the headings.) */
*****/
AFPVMoveTo(hPage, IN_2_U1440(4.5));
AFPHMoveTo(hPage, IN_2_U1440(.5));

return(0);
}

/*****
/* END A PAGE */
*****/
int end_page()
{
    int rc;
    char buffer[15];

    memset(buffer, 0x00, sizeof(buffer));

    AFPVMoveTo(hPage, IN_2_U1440(3));

    AFPHMoveTo(hPage, IN_2_U1440(7));

    sprintf(buffer, "  Page %d of  %d", PageNbr, NbrPages);

    AFPWriteString(hPage, buffer, -1, -1, LEFT_ALIGN, ' ', TRUE);

    rc = AFPEndPage(&hPage);

    checkForError(rc, "Error Ending a page\n");
```

```

return(0);
}

/*****
/* PROCESS PAGE INFORMATION */
*****/
int process_page_information()
{
int rc;

/*****
/* Create a new page */
*****/
PageNbr++;
create_page();

/*****
/* Create Table Definitions */
*****/
if (!TableIsDefined)
{
if (!create_new_table())
{
printf("\n create_new_table() Error!\n\n");
return(0);
}
}

/*****
/* Process Transactions */
*****/
if (!process_table_transactions())
{
printf("\n process_table_transactions() Error!\n\n");
TableProcessError = 1;
return(0);
}

/*****
/* End a page */
*****/
end_page();

rc = AFPEndGroup(hOutputDoc);

checkForError(rc, "Error Ending a Page Group\n");

return(1);
}

/*****
/* MAIN ROUTINE */
*****/
int main()
{
TBHANDLE toolboxSession;

int rc, cnt;

/*****
/* Start Toohbox Session */
*****/
rc = AFPBgnSession(&toolboxSession);

checkForError(rc, "Error Opening AFP Toolbox Session\n");

```

## C Sample

```
rc = AFPBgnDoc(toolboxSession, "TableProc Sample Test 1",
               outputDocComment, TypeOfOutput, szLogicalFileName,
               AfpRecBuffer, &hOutputDoc);

checkForError(rc, "Error Opening Output Document\n");

/*****
/* Define Fonts To Use */
*****/
printf("Defining Title font...\n");

rc = AFPDefineFontByAttr(outputCodepage,
                        "TIMES NEW ROMAN LATIN1", 140,
                        BOLD_WT, MEDIUM_WD, NORMAL_FS,
                        &titleFont);
checkForError(rc, "Error Defining Title Font\n");

printf("Defining Address font...\n");

rc = AFPDefineFontByAttr(outputCodepage,
                        "COURIER LATIN1", 120,
                        MEDIUM_WT, MEDIUM_WD, NORMAL_FS,
                        &addressFont);

checkForError(rc, "Error Defining Address Font\n");

printf("Defining Headings font...\n");

rc = AFPDefineFontByAttr(outputCodepage,
                        "TIMES NEW ROMAN LATIN1", 100,
                        MEDIUM_WT, MEDIUM_WD, NORMAL_FS,
                        &headingsFont);

checkForError(rc, "Error Defining Headings Font\n");

/*****
/* Process Page/Table Information */
*****/
TableIsDefined = ProcessedData1 = ProcessedData2 = PageNbr = 0;
TableProcessError = 0;

NbrPages = 2;

/*****
/* Begin group of pages */
*****/
rc = AFPBgnGroup(hOutputDoc, "Group Name 1");
checkForError(rc, "Error Beginning a Page Group\n");

for(cnt = 0; cnt < NbrPages; cnt++)
{
    if (!process_page_information())
        checkForError(RetCode,
                    "Error in process_page_information\n");
}

/*****
/* End group of pages */
*****/
rc = AFPEndGroup(hOutputDoc);
checkForError(rc, "Error Ending a Page Group\n");

/*****
/* Close Output Document Data File */
*****/
rc = AFPEndDoc(&hOutputDoc);
checkForError(rc, "Error Closing Output Document\n");
```

```

rc = AFPEndSession(&toolboxSession);
checkForError(rc, "Error Closing AFP Toolbox Session\n");

if (!TableProcessError)
    printf("\nTables Sample Test 1 Completed Successfully!\n");

return 0;
}
/*****
*
* IBM AFP Toolbox Table Processing
*
* Sample Source File TBLSUBR1.C
*
* Subrow Processing Test
*
*****/

#include stdio.h
#include stdlib.h
#include string.h
#include math.h
#include time.h
#ifdef __BORLANDC__
#include mem.h
#else
#include memory.h
#endif

#include "aocl.h"
#include "tblsubr1.h"

#ifdef AS400
static char szLogicalFileName[] = "/mylib.lib/afpout.file/tblsubr1.mbr";
static char OutputFile[] = "/mylib.lib/afpout.file/buffer.mbr";
char inputCodepage[] = "T1V10500";
#endif
#ifdef MVS
static char szLogicalFileName[] = "MYOUTPDD";
static char OutputFile[] = "tblsubr1.afp";
char inputCodepage[] = "T1V10500";
#else
char szLogicalFileName[] = "tblsubr1.afp";
char inputCodepage[] = "T1000850";
#endif

char outputCodepage[] = "T1V10500";
char outputDocComment[] = "AFP Toolbox Bank Statement "
    "Sample Applicaiton Output Document";
byte AfpRecBuffer[8192];
FILE* pFileOut;
OUTTYPE TypeOfOutput = TO_OUTPUT_FILE;

FontID titleFont, addressFont, headingsFont;
unsigned short far *ptitleFont, *paddressFont, *pheadingsFont;

FontID accountNameFont, transactionFont;

#ifdef __BORLANDC__
#define _USERENTRY
#endif

/*****/
/* ERROR INFO */
/*****/

```

## C Sample

```
void checkForError(int rc, char* message)
{
    if (rc != 0)
    {
        fprintf(stderr, message);
        fprintf(stderr, "AFP Toolbox return code %d\n\n", rc);
        exit(-1);
    }
}

/*****
/* CLEAR TRANS VARIABLES */
*****/
void ClearTransVars()
{
    memset(szCustName, 0x00, sizeof(szCustName));
    memset(szYearOfOrder1, 0x00, sizeof(szYearOfOrder1));
    memset(szYearOfOrder2, 0x00, sizeof(szYearOfOrder2));
    memset(szYearOfOrder3, 0x00, sizeof(szYearOfOrder3));
    memset(szGroupOrderSz1, 0x00, sizeof(szGroupOrderSz1));
    memset(szGroupOrderSz2, 0x00, sizeof(szGroupOrderSz2));
}

/*****
/* DEFINE TABLE HEADER FIELDS */
*****/
int define_table_hdr_flds()
{
    RetCode = 0;

    HdrFldArray[0] = &Hdrfld1
    HdrFldArray[1] = &Hdrfld2
    HdrFldArray[2] = &Hdrfld3
    HdrFldArray[3] = &Hdrfld4
    HdrFldArray[4] = &Hdrfld5
    HdrFldArray[5] = &Hdrfld6
    HdrFldArray[6] = &Hdrfld7
    HdrFldArray[7] = &Hdrfld8
    HdrFldArray[8] = &Hdrfld9
    HdrFldArray[9] = &Hdrfld10

    FormatOption = CENTER_ALIGN;
    //FormatOption = LEFT_ALIGN;

    //VertFormat = VERCENTER;
    VertFormat = VERTOP;
    TxtOrient = IOB90_T0;
    AlignPos = 0;

    // LeftMargin = 0.0;
    LeftMargin = 2.0;
    RightMargin = 0.0;

    LineSpace = -1;

    //ShadeIntense = 0.0;
    ShadeIntense = 16.0;

    /* Build Header Fields */
    for(Index = 0; Index < HdrFldCnt; Index++)
    {
        RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
                                LeftMargin, RightMargin, LineSpace, TxtOrient,
                                ShadeIntense, HdrFldArray[Index]);

        if (RetCode)
        {
            printf("\ndefine_table_hdr_flds Error1 !\n");
        }
    }
}
```

```

        printf("\nError return code: %d", RetCode);
        return(1);
    }
}

return(0);
}

/*****
/* DEFINE TABLE HEADER ROW */
*****/
int define_table_hdr_row()
{
    // Standard sz for table width is MM_2_U1440(180.0)
    // Equates to 10205 in 1440ths (i.e., BeginTable)

    /* Load global values */
    TopThick   = MM_2_U1440(.5);
    BotThick   = MM_2_U1440(.5);
    LeftThick  = MM_2_U1440(.5);
    RightThick = MM_2_U1440(.5);

    HdrColWidths[0] = 1800;
    HdrColWidths[1] = 1100;
    HdrColWidths[2] = 1100;
    HdrColWidths[3] = 1100;
    HdrColWidths[4] = 1250;
    HdrColWidths[5] = 1250;

    HdrRowArrange[0] = Hdrfld1;
    HdrRowArrange[1] = Hdrfld2;
    HdrRowArrange[2] = Hdrfld2;
    HdrRowArrange[3] = Hdrfld2;
    HdrRowArrange[4] = Hdrfld3;
    HdrRowArrange[5] = Hdrfld3;
    HdrRowArrange[6] = Hdrfld1;
    HdrRowArrange[7] = Hdrfld4;
    HdrRowArrange[8] = Hdrfld4;
    HdrRowArrange[9] = Hdrfld5;
    HdrRowArrange[10] = Hdrfld3;
    HdrRowArrange[11] = Hdrfld3;
    HdrRowArrange[12] = Hdrfld1;
    HdrRowArrange[13] = Hdrfld6;
    HdrRowArrange[14] = Hdrfld7;
    HdrRowArrange[15] = Hdrfld8;
    HdrRowArrange[16] = Hdrfld9;
    HdrRowArrange[17] = Hdrfld10;

    MinSubrowDepths[0] = 1.0;    /* MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH ? */

    RetCode= AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick,
                          BotThick, NbrofHdrColumns, NbrofHdrSubrows,
                          HdrRowArrange, HdrColWidths, &HdrRowId);

    if (RetCode)
    {
        printf("\ndefine_table_hdr_row Error !\n");
        printf("\nError return code: %d", RetCode);
        return(1);
    }

    return(0);
}

/*****
/* DEFINE TABLE TRANSACTION FIELDS */
*****/

```

## C Sample

```
int define_table_trans_flds()
{
    TrnFldArray[0] = &TrnFld1
    TrnFldArray[1] = &TrnFld2
    TrnFldArray[2] = &TrnFld3
    TrnFldArray[3] = &TrnFld4
    TrnFldArray[4] = &TrnFld5
    TrnFldArray[5] = &TrnFld6

    //FormatOption = CENTER_ALIGN;
    FormatOption = LEFT_ALIGN;
    //FormatOption = JUSTIFY_ALIGN;
    VertFormat = VERCENTER;
    TxtOrient = IOB90_T0;

    AlignPos = 0;
    ShadeIntense = 0.0;

    LeftMargin = 1.0;
    RightMargin = 1.0;

    /* Build Transaction Fields */
    for(Index = 0; Index < TrnFldCnt; Index++)
    {
        /* Check for field 3 to change margin and alignment position. */
        if (Index == 2)
        {
            LeftMargin = 0.0;
            AlignPos = 20;
        }

        RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
                                LeftMargin, RightMargin, LineSpace, TxtOrient,
                                ShadeIntense, TrnFldArray[Index]);

        if (RetCode)
        {
            printf("\ndefine_table_trans_flds Error !\n");
            printf("\nError return code: %d", RetCode);
            return(1);
        }
    }

    return(0);
}

/*****
/* DEFINE TABLE TRANSACTION ROWS */
*****/
int define_table_trans_rows()
{
    int iRowCnt;

    NbrofSubrows = 1;
    NbrofColumns = 6;

    ColumnWidths[0] = MM_2_U1440(40.0);
    ColumnWidths[1] = MM_2_U1440(40.0);
    ColumnWidths[2] = MM_2_U1440(40.0);
    ColumnWidths[3] = MM_2_U1440(40.0);
    ColumnWidths[4] = MM_2_U1440(40.0);
    ColumnWidths[5] = MM_2_U1440(40.0);

    MinSubrowDepths[0] = 1.0; /* MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH ? */

    RowArrange[0] = TrnFld1; /* MOVE FIELD TO AFP-ROW-ARRANGE */
    RowArrange[1] = TrnFld2;
    RowArrange[2] = TrnFld3;
```



```

RowArrange[3] = Trnflld4; /* MOVE FIELD TO AFP-ROW-ARRANGE */
RowArrange[4] = Trnflld5;
RowArrange[5] = Trnflld6;

for (iRowCnt = 0; iRowCnt < NBRTBLTRANSROWS; iRowCnt++)
{
    RetCode= AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick,
                          BotThick, NbrofColumns, NbrofSubrows,
                          RowArrange, ColumnWidths, &TrnsRows[iRowCnt]);

    if (RetCode)
    {
        printf("\ndefine_table_trans_rows Error !\n");
        printf("\nError return code: %d", RetCode);
        return(1);
    }
}

return(0);
}

/*****
/* DEFINE TABLE SUMMARY FIELDS */
*****/
int define_table_sum_flds()
{
    SumFldArray[0] = &Sumfld1
    SumFldArray[1] = &Sumfld2

    FormatOption = LEFT_ALIGN;
    //FormatOption = CENTER_ALIGN;
    VertFormat   = VERCENTER;
    TxtOrient    = IOB90_T0;

    AlignPos = 0;
    ShadeIntense = 18.0;

    LeftMargin  = 1.0;
    RightMargin = 1.0;

    /* Build Transaction Fields */
    for(Index = 0; Index < SumFldCnt; Index++)
    {
        /* Check for field 2 to change Margin and alignment position. */
        if (Index == 1)
        {
            LeftMargin  = 0.0;
            AlignPos = 20;
        }

        RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
                                LeftMargin, RightMargin, LineSpace, TxtOrient,
                                ShadeIntense, SumFldArray[Index]);

        if (RetCode)
        {
            printf("\ndefine_table_sum_flds Error !\n");
            printf("\nError return code: %d", RetCode);
            return(1);
        }
    }

    return(0);
}

/*****
/* DEFINE TABLE SUMMARY ROW */
*****/
int define_table_sum_row()

```

## C Sample

```
{
NbrofSubrows = 1;
NbrofColumns = 2;

ColumnWidths[0] = MM_2_U1440(90.0);
ColumnWidths[1] = MM_2_U1440(90.0);
ColumnWidths[2] = 0.0;

MinSubrowDepths[0] = 1.0; /* MOVE AFP-DEFAULT TO AFP-SUBROW-DEPTH ? */

RowArrange[0] = Sumfld1; /* MOVE FIELD TO AFP-ROW-ARRANGE */
RowArrange[1] = Sumfld2;
RowArrange[2] = 0x00;

RetCode= AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick, BotThick,
                      NbrofColumns, NbrofSubrows, RowArrange,
                      ColumnWidths, &SumRowId);

if (RetCode)
{
    printf("\ndefine_table_sum_row Error !\n");
    printf("\nError return code: %d", RetCode);
    return(1);
}

return(0);
}

/*****
/* WRITE TABLE HEADER ROW */
*****/
int write_table_hdr_row()
{
    /* Begin Header Row */
    RetCode = AFPBeginRow(hTbl, HdrRowId);

    if (RetCode)
        return(1);

    /* Begin Header Field 1 */
    RetCode = AFPBeginField(hTbl, Hdrfld1);

    if (RetCode)
        return(1);

    /* Put Text in Header Field 1 */
    memcpy(szTextstring, NULL, sizeof(szTextstring));
    strcpy(szTextstring, "Customer Name");
    /* Set alignment CENTER here. */

    //pheadingsFont = &headingsFont
    RetCode = AFPPutTextInTableFld(hTbl, Hdrfld1,
                                   szTextstring, headingsFont);
    /*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/
    if (RetCode)
        return(1);

    /* End Header Field 1 */
    RetCode = AFPEndField(hTbl);
    if (RetCode)
        return(1);

    /* Begin Header Field 2 */
    RetCode = AFPBeginField(hTbl, Hdrfld2);
    if (RetCode)
```

```

    return(1);

/* Put Text in Header Field 2 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Year of Orders");
/* Set alignment CENTER here. */

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld2,
                               szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/
if (RetCode)
    return(1);

/* End Header Field 2 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(1);

/* Begin Header Field 3 */
RetCode = AFPBeginField(hTbl, Hdrfld3);
if (RetCode)
    return(1);

/* Put Text in Header Field 3 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Grouping of order size");
/* Set alignment CENTER here. */

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld3,
                               szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/
if (RetCode)
    return(1);
/* End Header Field 3 */
RetCode = AFPEndField(hTbl);

/* Begin Header Field 4 */
RetCode = AFPBeginField(hTbl, Hdrfld4);
if (RetCode)
    return(1);

/* Put Text in Header Field 4 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "20th Century");
/* Set alignment CENTER here. */

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld4,
                               szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/
if (RetCode)
    return(1);
/* End Header Field 4 */
RetCode = AFPEndField(hTbl);

/* Begin Header Field 5 */
RetCode = AFPBeginField(hTbl, Hdrfld5);
if (RetCode)
    return(1);

/* Put Text in Header Field 5 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "21st");
/* Set alignment CENTER here. */

```

## C Sample

```
RetCode = AFPPutTextInTableFld(hTbl, Hdrfld5,
                               szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/
if (RetCode)
    return(1);
/* End Header Field 5 */
RetCode = AFPEndField(hTbl);

/* Begin Header Field 6 */
RetCode = AFPBeginField(hTbl, Hdrfld6);
if (RetCode)
    return(1);

/* Put Text in Header Field 6 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "1998");
/* Set alignment CENTER here. */

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld6,
                               szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/
if (RetCode)
    return(1);

/* End Header Field 6 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(1);

/* Begin Header Field 7 */
RetCode = AFPBeginField(hTbl, Hdrfld7);
if (RetCode)
    return(1);

/* Put Text in Header Field 7 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "1999");
/* Set alignment CENTER here. */

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld7,
                               szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/
if (RetCode)
    return(1);
/* End Header Field 7 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(1);

/* Begin Header Field 8 */
RetCode = AFPBeginField(hTbl, Hdrfld8);
if (RetCode)
    return(1);

/* Put Text in Header Field 8 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "2000");
/* Set alignment CENTER here. */

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld8,
                               szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/
if (RetCode)
    return(1);
/* End Header Field 8 */
```

```

RetCode = AFPEndField(hTbl);
if (RetCode)
    return(1);

/* Begin Header Field 9 */
RetCode = AFPBeginField(hTbl, Hdrfld9);
if (RetCode)
    return(1);

/* Put Text in Header Field 9 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "0-$50k");
/* Set alignment CENTER here. */

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld9,
                                szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/
if (RetCode)
    return(1);
/* End Header Field 9 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(1);

/* Begin Header Field 10 */
RetCode = AFPBeginField(hTbl, Hdrfld10);
if (RetCode)
    return(1);

/* Put Text in Header Field 10 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, ">$50k");
/* Set alignment CENTER here. */

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld10,
                                szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/
if (RetCode)
    return(1);
/* End Header Field 10 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(1);

/* End Header Row */
RetCode = AFPEndRow(hTbl, HdrRowId, &CurrTableDepth);

if (RetCode)
    return(1);

return(0);
}

/*****
/* WRITE TABLE TRANSACTION ROW */
*****/
int write_table_trans_row(unsigned int iRowId,
    char* szName, char* szYrOrder1, char *szYrOrder2,
    char* szYrOrder3, char* szGrpOrderSz1, char* szGrpOrderSz2)
{
/* Begin Transaction Row */
RetCode = AFPBeginRow(hTbl, iRowId);

if (RetCode)
    return(RetCode);

```

## C Sample

```
/* Write Transaction Field 1 */
RetCode = AFPBeginField(hTbl, TrnFld1);

if (RetCode)
    return(RetCode);

/* Customer Name */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, szName);

RetCode = AFPPutTextInTableFld(hTbl, TrnFld1, szTextstring,
                               headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 1 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Begin Transaction Field 2 */
RetCode = AFPBeginField(hTbl, TrnFld2);
if (RetCode)
    return(RetCode);

/* Year of Orders 1 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, szYrOrder1);

RetCode = AFPPutTextInTableFld(hTbl, TrnFld2,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 2 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Begin Transaction Field 3 */
RetCode = AFPBeginField(hTbl, TrnFld3);
if (RetCode)
    return(RetCode);

/* Year of Orders 2 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, szYrOrder2);

RetCode = AFPPutTextInTableFld(hTbl, TrnFld3,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 3 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Begin Transaction Field 4 */
RetCode = AFPBeginField(hTbl, TrnFld4);
if (RetCode)
    return(RetCode);

/* Year of Orders 3 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, szYrOrder3);
```

```

RetCode = AFPPutTextInTableFld(hTbl, TrnFld4,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 4 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Begin Transaction Field 5 */
RetCode = AFPBeginField(hTbl, TrnFld5);
if (RetCode)
    return(RetCode);

/* Grouping of Order size 1 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, szGrpOrderSz1);

RetCode = AFPPutTextInTableFld(hTbl, TrnFld5,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 5 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Begin Transaction Field 6 */
RetCode = AFPBeginField(hTbl, TrnFld6);
if (RetCode)
    return(RetCode);

/* Grouping of Order size 2 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, szGrpOrderSz2);

RetCode = AFPPutTextInTableFld(hTbl, TrnFld6,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 6 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* End Transaction Row */
RetCode = AFPEndRow(hTbl, iRowId, &CurrTableDepth);
if (RetCode)
    return(RetCode);

return(0);
}

/*****
/* WRITE TABLE SUMMARY ROW */
*****/
int write_table_sum_row()
{
    /* Begin Summary Row */
    RetCode = AFPBeginRow(hTbl, SumRowId);
    if (RetCode)
        return(RetCode);

```

## C Sample

```
/* Begin Summary Field 1 */
RetCode = AFPBeginField(hTbl, Sumfld1);
if (RetCode)
    return(RetCode);

/* Summary Amount */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Total Amount");

RetCode = AFPPutTextInTableFld(hTbl, Sumfld1,
                               szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/
if (RetCode)
    return(RetCode);

/* End Summary Field 1 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Write Summary Field 2 */
RetCode = AFPBeginField(hTbl, Sumfld2);
if (RetCode)
    return(RetCode);

/* Summary Total */
memcpy(szTextstring, NULL, sizeof(szTextstring));
sprintf(szTextstring, "%s%.2f", "$", fTransTotal);

RetCode = AFPPutTextInTableFld(hTbl, Sumfld2,
                               szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &addressFont);*/

if (RetCode)
    return(RetCode);

/* End Transaction Field 2 */
RetCode = AFPEndField(hTbl);

if (RetCode)
    return(RetCode);

/* End Summary Row */
RetCode = AFPEndRow(hTbl, SumRowId, &CurrTableDepth);

if (RetCode)
    return(RetCode);

return(0);
}

/*****
/* PROCESS ROWS(1ST GROUP OF TRANSACTIONS */
*****/
int process_transaction_data1()
{
    ClearTransVars();
    strcpy(szCustName, "Allied");
    strcpy(szYearOfOrder1, "12 units");
    strcpy(szYearOfOrder2, "11 units");
    strcpy(szYearOfOrder3, "7 units");
    strcpy(szGroupOrderSz1, "24 units");
    strcpy(szGroupOrderSz2, "6 units");

    RetCode = write_table_trans_row(TrnsRows[0], szCustName,
                                   szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
                                   szGroupOrderSz1, szGroupOrderSz2);
```



```

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Petro");
strcpy(szYearOfOrder1, "10 units");
strcpy(szYearOfOrder2, "10 units");
strcpy(szYearOfOrder3, "7 units");
strcpy(szGroupOrderSz1, "20 units");
strcpy(szGroupOrderSz2, "7 units");

RetCode = write_table_trans_row(TrnsRows[1], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Cardinal");
strcpy(szYearOfOrder1, "5 units");
strcpy(szYearOfOrder2, "7 units");
strcpy(szYearOfOrder3, "7 units");
strcpy(szGroupOrderSz1, "9 units");
strcpy(szGroupOrderSz2, "11 units");

RetCode = write_table_trans_row(TrnsRows[2], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Ashcroft");
strcpy(szYearOfOrder1, "2 units");
strcpy(szYearOfOrder2, "3 units");
strcpy(szYearOfOrder3, "4 units");
strcpy(szGroupOrderSz1, "5 units");
strcpy(szGroupOrderSz2, "4 units");

RetCode = write_table_trans_row(TrnsRows[3], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Smith-Kein");
strcpy(szYearOfOrder1, "12 units");
strcpy(szYearOfOrder2, "13 units");
strcpy(szYearOfOrder3, "20 units");
strcpy(szGroupOrderSz1, "5 units");
strcpy(szGroupOrderSz2, "40 units");

RetCode = write_table_trans_row(TrnsRows[4], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Thompson");
strcpy(szYearOfOrder1, "6 units");

```

## C Sample

```
strcpy(szYearOfOrder2, "5 units");
strcpy(szYearOfOrder3, "4 units");
strcpy(szGroupOrderSz1, "5 units");
strcpy(szGroupOrderSz2, "10 units");

RetCode = write_table_trans_row(TrnsRows[5], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Johnson");
strcpy(szYearOfOrder1, "2 units");
strcpy(szYearOfOrder2, "3 units");
strcpy(szYearOfOrder3, "4 units");
strcpy(szGroupOrderSz1, "4 units");
strcpy(szGroupOrderSz2, "5 units");

RetCode = write_table_trans_row(TrnsRows[6], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Altman");
strcpy(szYearOfOrder1, "1 units");
strcpy(szYearOfOrder2, "5 units");
strcpy(szYearOfOrder3, "8 units");
strcpy(szGroupOrderSz1, "4 units");
strcpy(szGroupOrderSz2, "10 units");

RetCode = write_table_trans_row(TrnsRows[7], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Beechman");
strcpy(szYearOfOrder1, "11 units");
strcpy(szYearOfOrder2, "15 units");
strcpy(szYearOfOrder3, "18 units");
strcpy(szGroupOrderSz1, "40 units");
strcpy(szGroupOrderSz2, "4 units");

RetCode = write_table_trans_row(TrnsRows[8], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Sinclair");
strcpy(szYearOfOrder1, "10 units");
strcpy(szYearOfOrder2, "10 units");
strcpy(szYearOfOrder3, "18 units");
strcpy(szGroupOrderSz1, "8 units");
strcpy(szGroupOrderSz2, "30 units");

RetCode = write_table_trans_row(TrnsRows[9], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
```

```

    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Edison");
strcpy(szYearOfOrder1, "6 units");
strcpy(szYearOfOrder2, "7 units");
strcpy(szYearOfOrder3, "11 units");
strcpy(szGroupOrderSz1, "4 units");
strcpy(szGroupOrderSz2, "20 units");

RetCode = write_table_trans_row(TrnsRows[10], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Powell");
strcpy(szYearOfOrder1, "2 units");
strcpy(szYearOfOrder2, "2 units");
strcpy(szYearOfOrder3, "12 units");
strcpy(szGroupOrderSz1, "4 units");
strcpy(szGroupOrderSz2, "10 units");

RetCode = write_table_trans_row(TrnsRows[11], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Dawson");
strcpy(szYearOfOrder1, "12 units");
strcpy(szYearOfOrder2, "22 units");
strcpy(szYearOfOrder3, "12 units");
strcpy(szGroupOrderSz1, "24 units");
strcpy(szGroupOrderSz2, "22 units");

RetCode = write_table_trans_row(TrnsRows[12], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

ClearTransVars();
strcpy(szCustName, "Edwards");
strcpy(szYearOfOrder1, "2 units");
strcpy(szYearOfOrder2, "2 units");
strcpy(szYearOfOrder3, "2 units");
strcpy(szGroupOrderSz1, "3 units");
strcpy(szGroupOrderSz2, "3 units");

RetCode = write_table_trans_row(TrnsRows[13], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode);

/*
ClearTransVars();

```

## C Sample

```
strcpy(szCustName, "Baxter");
strcpy(szYearOfOrder1, "100 units");
strcpy(szYearOfOrder2, "200 units");
strcpy(szYearOfOrder3, "300 units");
strcpy(szGroupOrderSz1, "400 units");
strcpy(szGroupOrderSz2, "200 units");

RetCode = write_table_trans_row(TrnsRows[14], szCustName,
    szYearOfOrder1, szYearOfOrder2, szYearOfOrder3,
    szGroupOrderSz1, szGroupOrderSz2);

if (RetCode)
    return(RetCode); /*

/* Set process switch */
ProcessedData1 = 1;

return(0);
}

/*****
/* PROCESS ROWS(2ND GROUP OF TRANSACTIONS */
*****/
int process_transaction_data2()
{

/* Set processed switch */
ProcessedData2 = 1;

return(0);
}

/*****
/* PROCESS TABLE TRANSACTIONS */
*****/
int process_table_transactions()
{
char szDate[75];
char szTrans[75];
float fValue = 0.0;

AFPSetPos(hPage, MM_2_U1440(20), ABSOLUTE_POS,
    MM_2_U1440(80), ABSOLUTE_POS);

AFPVMoveTo(hPage, IN_2_U1440(3.5));

/* Begin Table */
TopThick = .5;
BotThick = .5;
LeftThick = .5;
RightThick = .5;

RetCode = AFPBeginTable(hPage, MM_2_U1440(130.0),
    MM_2_U1440(80.0), 0, MM_2_U1440(.5), MM_2_U1440(.5),
    MM_2_U1440(.5), MM_2_U1440(.5), &hTbl);

if (RetCode)
    return(0);

/* Write Table Header Row */
RetCode = write_table_hdr_row();

if (RetCode)
    return(0);

/* Start Writing Table Transaction Rows */
if (!ProcessedData1)
```

```

{
    RetCode = process_transaction_data1();

    checkForError(RetCode, "process_transaction_data1\n");
}
/*
else
if (!ProcessedData2)
{
    RetCode = process_transaction_data2();

    * Page/Table Depth Error *
    if (RetCode == 37)
    {
        memcpy(szTextstring, NULL, sizeof(szTextstring));
        sprintf(szTextstring, "%s%d",
            "\n Table Depth Error(Data 2) On Page ", PageNbr);
        printf(szTextstring);

        return(0);
    }

    * Write Table Summary Row *
    RetCode = write_table_sum_row();

    * Page/Table Depth Error *
    if (RetCode == 37)
    {
        memcpy(szTextstring, NULL, sizeof(szTextstring));
        sprintf(szTextstring, "%s%d",
            "\n Table Depth Error(Summary Row) On Page ", PageNbr);
        printf(szTextstring);
        return(0);
    }
}
*/

/* End Table */
RetCode = AFPEndTable(&hTbl, &CurrTableDepth);

if (RetCode)
    return(0);

return(1);
}

/*****/
/* CREATE NEW TABLE */
/*****/
int create_new_table()
{
    /*****/
    /* Define Header Fields */
    /*****/
    RetCode = define_table_hdr_flds();
    if (RetCode)
        return(0);

    /*****/
    /* Define Header Row */
    /*****/
    RetCode = define_table_hdr_row();
    if (RetCode)
        return(0);

    /*****/
    /* Define Transaction Fields */

```

## C Sample

```

/*****/
RetCode = define_table_trans_flds();
if (RetCode)
    return(0);

/*****/
/* Define Transaction Row */
/*****/
RetCode = define_table_trans_rows();
if (RetCode)
    return(0);

/*****/
/* Define Summary Fields */
/*****/
RetCode = define_table_sum_flds();
if (RetCode)
    return(0);

/*****/
/* Define Summary Row */
/*****/
RetCode = define_table_sum_row();
if (RetCode)
    return(0);

/*****/
/* Table Definitions Successful ! */
/*****/
TableIsDefined = 1;

return(1);
}

/*****/
/* CREATE NEW PAGE */
/*****/
int create_page()
{
    int rc;
    long lpos;

    char buffer[15];
    char* datestr;

    /*****/
    /* SET GLOBAL PAGE HANDLE */
    /*****/
    hPage = NULL;

    /*****/
    /* Begin a page. */
    /*****/
    rc = AFPBgnPage(hOutputDoc, "Table Page", &hPage);

    checkForError(rc, "Error Creating a Page\n");

    /*****/
    /* Include Logo Page Segment in upper left corner. */
    /*****/
    AFPHMoveTo(hPage, IN_2_U1440(.5));
    AFPVMoveTo(hPage, IN_2_U1440(.5));
    /*AFPInclPSeg(hPage, "WSLLOGO", TRUE);*/

    /*****/
    /* Set Font, move and write company address. */
    /*****/

```

```

AFPSetFont(hPage, titleFont);

AFPVMoveTo(hPage, IN_2_U1440(.75));

AFPHMoveTo(hPage, IN_2_U1440(3.5));

AFPWriteString(hPage, "IBM Printing Systems", -1, -1, LEFT_ALIGN, ' ', TRUE);

AFPNextLine(hPage);

AFPWriteString(hPage, "6300 Diagonal Hwy.", -1, -1, LEFT_ALIGN, ' ', TRUE);

AFPNextLine(hPage);

AFPWriteString(hPage, "Boulder, CO 80301", -1, -1, LEFT_ALIGN, ' ', TRUE);

AFPSetFont(hPage, headingsFont);

AFPHMoveTo(hPage, IN_2_U1440(.5));

AFPVMoveTo(hPage, IN_2_U1440(2.5));

AFPWriteString(hPage, "Table Processing Test", -1, -1, LEFT_ALIGN,
    ' ', TRUE);

AFPNextLine(hPage);

AFPWriteString(hPage, "Developer: Brian Smith", -1, -1, LEFT_ALIGN,
    ' ', TRUE);

AFPNextLine(hPage);
AFPNextLine(hPage);
AFPNextLine(hPage);
AFPNextLine(hPage);

/*****
/* Move under address, print statement dates. */
*****/
AFPHMoveTo(hPage, IN_2_U1440(3.5));
AFPVMoveTo(hPage, IN_2_U1440(2.5));

AFPWriteString(hPage, "Statement From: ", -1, -1, RIGHT_ALIGN, ' ', TRUE);
AFPWriteString(hPage, "July 1, 2000", -1, -1, LEFT_ALIGN, ' ', TRUE);
AFPNextLine(hPage);

AFPHMoveTo(hPage, IN_2_U1440(3.2));

AFPWriteString(hPage, "To: July 26, 2000", -1, -1, LEFT_ALIGN, ' ', TRUE);

/*****
/* Add Bottom Disclaimer to Document */
*****/
AFPVMoveTo(hPage, IN_2_U1440(10.55));
AFPHMoveTo(hPage, IN_2_U1440(8.5/2.0));

AFPWriteString(hPage, "IBM AFP Toolbox Sample Bank "
    "Statement Application Output",
    -1, -1, CENTER_ALIGN, ' ', TRUE);

/*****
/* Move current position (under the headings.) */
*****/
AFPVMoveTo(hPage, IN_2_U1440(4.5));
AFPHMoveTo(hPage, IN_2_U1440(.5));

return(0);
}

```

## C Sample

```
/* ***** */
/* END A PAGE */
/* ***** */
int end_page()
{
    int rc;
    char buffer[15];

    memset(buffer, 0x00, sizeof(buffer));

    AFPVMoveTo(hPage, IN_2_U1440(3));

    AFPHMoveTo(hPage, IN_2_U1440(7));

    sprintf(buffer, " Page %d of %d", PageNbr, NbrPages);

    AFPWriteString(hPage, buffer, -1, -1, LEFT_ALIGN, ' ', TRUE);

    rc = AFPEndPage(&hPage);

    checkForError(rc, "Error Ending a page\n");

    return(0);
}

/* ***** */
/* PROCESS PAGE INFORMATION */
/* ***** */
int process_page_information()
{
    int rc;

    /* ***** */
    /* Create a new page */
    /* ***** */
    PageNbr++;
    create_page();

    /* ***** */
    /* Create Table Definitions */
    /* ***** */
    if (!TableIsDefined)
    {
        if (!create_new_table())
        {
            printf("\n create_new_table() Error!\n\n");
            return(0);
        }
    }

    /* ***** */
    /* Process Transactions */
    /* ***** */
    if (!process_table_transactions())
    {
        printf("\n process_table_transactions() Error!\n\n");
        TableProcessError = 1;
        return(0);
    }

    /* ***** */
    /* End a page */
    /* ***** */
    end_page();

    rc = AFPEndGroup(hOutputDoc);
```



```

checkForError(rc, "Error Ending a Page Group\n");

return(1);
}

/*****
/* MAIN ROUTINE */
*****/
int main()
{
    TBHANDLE toolboxSession;

    int rc, cnt;

    /*****
    /* Start Toohbox Session */
    *****/
    rc = AFPBgnSession(&toolboxSession);

    checkForError(rc, "Error Opening AFP Toolbox Session\n");

    rc = AFPBgnDoc(toolboxSession, "TABERR4E",
        outputDocComment, TypeOfOutput, szLogicalFileName,
        AfpRecBuffer, &hOutputDoc);

    checkForError(rc, "Error Opening Output Document\n");

    /*****
    /* Define Fonts To Use */
    *****/
    printf("Defining Title font...\n");

    rc = AFPDefineFontByAttr(outputCodepage,
        "TIMES NEW ROMAN LATIN1", 140,
        BOLD_WT, MEDIUM_WD, NORMAL_FS,
        &titleFont);
    checkForError(rc, "Error Defining Title Font\n");

    printf("Defining Address font...\n");

    rc = AFPDefineFontByAttr(outputCodepage,
        "COURIER LATIN1", 120,
        MEDIUM_WT, MEDIUM_WD, NORMAL_FS,
        &addressFont);

    checkForError(rc, "Error Defining Address Font\n");

    printf("Defining Headings font...\n");

    rc = AFPDefineFontByAttr(outputCodepage,
        "TIMES NEW ROMAN LATIN1", 100,
        MEDIUM_WT, MEDIUM_WD, NORMAL_FS,
        &headingsFont);

    checkForError(rc, "Error Defining Headings Font\n");

    /*****
    /* Process Page/Table Information */
    *****/
    TableIsDefined = ProcessedData1 = ProcessedData2 = PageNbr = 0;
    TableProcessError = 0;

    NbrPages = 1;

    /*****
    /* Begin group of pages */

```

## C Sample

```
/*
/*****
rc = AFPBgnGroup(hOutputDoc, "Group Name 1");
checkForError(rc, "Error Beginning a Page Group\n");

for(cnt = 0; cnt < NbrPages; cnt++)
{
    process_page_information();
}

/*****
/* End group of pages */
/*****
rc = AFPEndGroup(hOutputDoc);
checkForError(rc, "Error Ending a Page Group\n");

/*****
/* Close Output Document Data File */
/*****
rc = AFPEndDoc(&hOutputDoc);
checkForError(rc, "Error Closing Output Document\n");

rc = AFPEndSession(&toolboxSession);
checkForError(rc, "Error Closing AFP Toolbox Session\n");

if (!TableProcessError)
    printf("\nTables Sample Test TBLSUBR1.C Completed Successfully!\n");

return 0;
}
*/

```

---

## Sample in C++

```
/*
/*****
*
* IBM AFP Toolbox Table Processing
*
* This sample demonstrates arranged fields in a row
* with different vertical and horizontal formatting.
*
*****/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#ifdef __BORLANDC__
#include <mem.h>
#else
#include <memory.h>
#endif

#include "aocl.h"
#include "tblsubr1.h"

#ifdef AS400
static char szLogicalFileName[] = "/mylib.lib/afpout.file/tblsubr1.mbr";
char inputCodepage[] = "T1V10500";
#endif
#ifdef MVS
static char szLogicalFileName[] = "FILEOUT";
char inputCodepage[] = "T1V10500";
#else
char szLogicalFileName[] = "tblsubr1.afp";
char inputCodepage[] = "T1000850";
#endif

```

```

char outputCodepage[] = "T1V10500";
char outputDocComment[] = "AFP Toolbox Table Test TBLCTST2";
byte AfpRecBuffer[8192];
OUTTYPE TypeOfOutput = TO_OUTPUT_FILE;

FontID transFont, headingsFont;
unsigned short far *ptransFont, *pheadingsFont;

short jlsfld1, jlsfld2, jlsfld3;
short jlsfld4, jlsfld5, jlsfld6;
short jlsfld7, jlsfld8;
short* JLSFldArray[8];

/*****/
/* ERROR INFO */
/*****/
void checkForError(int rc, char* message)
{
    if (rc != 0)
    {
        fprintf(stderr, message);
        fprintf(stderr, "AFP Toolbox return code %d\n", rc);
        exit(-1);
    }
    else
    {
        fprintf(stderr, "Call completed OK: %s\n", message);
    }
}

/*****/
/* DEFINE TABLE HEADER FIELDS */
/*****/
int define_table_hdr_flds()
{
    RetCode = 0;

    HdrFldArray[0] = &Hdrfld1
    HdrFldArray[1] = &Hdrfld2
    HdrFldArray[2] = &Hdrfld3

    VertFormat = VERCENTER;
    TxtOrient = IOB90_TO;
    AlignPos = 0;

    LeftMargin = MM_2_U1440(2.0);
    RightMargin = 0.0;

    LineSpace = -1;

    ShadeIntense = 18.0;

    /* Build Header Fields */
    FormatOption = LEFT_ALIGN;
    RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
        LeftMargin, RightMargin, LineSpace, TxtOrient,
        STANDARD_SHADE, ShadeIntense, HdrFldArray[0]);
    checkForError(RetCode, "Define Field\n");
    FormatOption = CENTER_ALIGN;
    RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
        LeftMargin, RightMargin, LineSpace, TxtOrient,
        STANDARD_SHADE, ShadeIntense, HdrFldArray[1]);
    checkForError(RetCode, "Define Field\n");
    FormatOption = RIGHT_ALIGN;
    RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
        LeftMargin, RightMargin, LineSpace, TxtOrient,
        STANDARD_SHADE, ShadeIntense, HdrFldArray[2]);

```

## C++ Sample

```
    checkForError(RetCode,"Define Field\n");

return(0);
}

/*****
/* DEFINE TABLE HEADER ROW */
*****/
int define_table_hdr_row()
{

/* Load global values */
    TopThick   = MM_2_U1440(.5);
    BotThick   = MM_2_U1440(.5);

HdrColWidths[0] = MM_2_U1440(40);
HdrColWidths[1] = MM_2_U1440(80);
HdrColWidths[2] = MM_2_U1440(30);

HdrRowArrange[0] = Hdrfld1;
HdrRowArrange[1] = Hdrfld2;
HdrRowArrange[2] = Hdrfld3;
NbrofHdrColumns = 3;
NbrofHdrSubrows = 1;

MinSubrowDepths[0] = IN_2_U1440(.5);

RetCode= AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick,
                      BotThick, NbrofHdrColumns, NbrofHdrSubrows,
                      &HdrRowArrange[0], HdrColWidths, &HdrRowId);

checkForError(RetCode,"Define Row\n");
return(0);
}

/*****
/* DEFINE TABLE TRANSACTION FIELDS */
*****/
int define_table_trans_flds()
{
JLSFldArray[0] = &jlsfld1
JLSFldArray[1] = &jlsfld2
JLSFldArray[2] = &jlsfld3
JLSFldArray[3] = &jlsfld4
JLSFldArray[4] = &jlsfld5
JLSFldArray[5] = &jlsfld6
JLSFldArray[6] = &jlsfld7
JLSFldArray[7] = &jlsfld8

TxtOrient      = I0B90_T0;

AlignPos = 0;
ShadeIntense = 0.0;

/* Build Transaction Fields */
    FormatOption = LEFT_ALIGN;
    VertFormat   = VERCENTER;
    RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
                             LeftMargin, RightMargin, LineSpace, TxtOrient,
                             STANDARD_SHADE, ShadeIntense, JLSFldArray[0]);
    checkForError(RetCode,"Define Field\n");
    VertFormat   = VERTOP;
    RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
                             LeftMargin, RightMargin, LineSpace, TxtOrient,
                             STANDARD_SHADE, ShadeIntense, JLSFldArray[1]);
```

```

checkForError(RetCode,"Define Field\n");
LeftMargin = 0.0;
AlignPos = 20;

FormatOption = RIGHT_ALIGN;
VertFormat = VERBOTTOM;
RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
                        LeftMargin, RightMargin, LineSpace, TxtOrient,
                        STANDARD_SHADE, ShadeIntense, JLSFldArray[2]);
checkForError(RetCode,"Define Field\n");
FormatOption = LEFT_ALIGN;
VertFormat = VERCENTER;
RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
                        LeftMargin, RightMargin, LineSpace, TxtOrient,
                        STANDARD_SHADE, ShadeIntense, JLSFldArray[3]);
VertFormat = VERBOTTOM;
RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
                        LeftMargin, RightMargin, LineSpace, TxtOrient,
                        STANDARD_SHADE, ShadeIntense, JLSFldArray[4]);

LeftMargin = 0.0;
AlignPos = 20;

FormatOption = RIGHT_ALIGN;
VertFormat = VERTOP;
RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
                        LeftMargin, RightMargin, LineSpace, TxtOrient,
                        STANDARD_SHADE, ShadeIntense, JLSFldArray[5]);
checkForError(RetCode,"Define Field\n");
FormatOption = CENTER_ALIGN;
RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
                        LeftMargin, RightMargin, LineSpace, TxtOrient,
                        STANDARD_SHADE, ShadeIntense, JLSFldArray[6]);
checkForError(RetCode,"Define Field\n");
RetCode = AFPDefineField(hOutputDoc, FormatOption, AlignPos, VertFormat,
                        LeftMargin, RightMargin, LineSpace, TxtOrient,
                        STANDARD_SHADE, ShadeIntense, JLSFldArray[7]);
checkForError(RetCode,"Define Field\n");

return(0);
}

/*****
/* DEFINE TABLE TRANSACTION ROWS */
*****/
int define_table_trans_rows()
{

NbrofSubrows = 4;
NbrofColumns = 3;
ColumnWidths[0] = MM_2_U1440(45.0);
ColumnWidths[1] = MM_2_U1440(70.0);
ColumnWidths[2] = MM_2_U1440(35.0);

MinSubrowDepths[0] = IN_2_U1440(1);
MinSubrowDepths[1] = IN_2_U1440(1);
MinSubrowDepths[2] = IN_2_U1440(1);
MinSubrowDepths[3] = IN_2_U1440(1);

RowArrange[0] = jlsfld1; /* MOVE FIELD TO AFP-ROW-ARRANGE */
RowArrange[1] = jlsfld2;
RowArrange[2] = jlsfld3;
RowArrange[3] = jlsfld1; /* MOVE FIELD TO AFP-ROW-ARRANGE */
RowArrange[4] = jlsfld4;
RowArrange[5] = jlsfld3;
RowArrange[6] = jlsfld5; /* MOVE FIELD TO AFP-ROW-ARRANGE */
RowArrange[7] = jlsfld4;
RowArrange[8] = jlsfld6;

```

## C++ Sample

```
RowArrange[9] = jlsfld5; /* MOVE FIELD TO AFP-ROW-ARRANGE */
RowArrange[10] = jlsfld7;
RowArrange[11] = jlsfld8;

RetCode= AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick,
                      BotThick, NbrofColumns, NbrofSubrows,
                      &RowArrange[0], ColumnWidths, &TrnsRows[0]);
checkForError(RetCode,"Define Row\n");
RetCode= AFPDefineRow(hOutputDoc, MinSubrowDepths, TopThick,
                      BotThick, NbrofColumns, NbrofSubrows,
                      &RowArrange[0], ColumnWidths, &TrnsRows[1]);
checkForError(RetCode,"Define Row\n");

return(0);
}

/*****
/* WRITE TABLE HEADER ROW */
*****/
int write_table_hdr_row()
{
/* Begin Header Row */
RetCode = AFPBeginRow(hTbl, HdrRowId);
checkForError(RetCode,"Begin Row\n");

/* Begin Header Field 1 */
RetCode = AFPBeginField(hTbl, Hdrfld1);
checkForError(RetCode,"Begin Field Hdrfld1\n");

/* Put Text in Header Field 1 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Left/Vercenter");

//pheadingsFont = &headingsFont
RetCode = AFPPutTextInTableFld(hTbl, Hdrfld1,
                              szTextstring, headingsFont);
checkForError(RetCode,"PutText Field Hdrfld1\n");

/* End Header Field 1 */
RetCode = AFPEndField(hTbl);
checkForError(RetCode,"End Field Hdrfld1\n");

/* Begin Header Field 2 */
RetCode = AFPBeginField(hTbl, Hdrfld2);
checkForError(RetCode,"Bgn Field Hdrfld2\n");

/* Put Text in Header Field 2 */
memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Centered/Vercenter");

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld2,
                              szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &transFont);*/
checkForError(RetCode,"PutText Field Hdrfld2\n");

/* End Header Field 2 */
RetCode = AFPEndField(hTbl);
checkForError(RetCode,"End Field Hdrfld2\n");

/* Begin Header Field 3 */
RetCode = AFPBeginField(hTbl, Hdrfld3);
checkForError(RetCode,"Bgn Field Hdrfld3\n");

/* Put Text in Header Field 3 */
```

```

memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Right/Vercenter");

RetCode = AFPPutTextInTableFld(hTbl, Hdrfld3,
                               szTextstring, headingsFont);
/*RetCode = AFPPutTextInTableFld(hTbl, szTextstring, &transFont);*/
checkForError(RetCode,"Put Field Hdrfld3\n");
/* End Header Field 3 */
RetCode = AFPEndField(hTbl);

/* End Header Row */
RetCode = AFPEndRow(hTbl, HdrRowId, &CurrTableDepth);
checkForError(RetCode,"End Field Hdrfld3\n");

return(0);
}

/*****
/* WRITE TABLE TRANSACTION ROW */
*****/
int write_table_trans_row()
{
/* Begin Transaction Row */
RetCode = AFPBeginRow(hTbl, TrnsRows[0]);
checkForError(RetCode,"Begin Row Trns0\n");

if (RetCode)
    return(RetCode);

/* Write Transaction Field 1 */
RetCode = AFPBeginField(hTbl, jlsfld1);
checkForError(RetCode,"Begin Field jlsfld1\n");

if (RetCode)
    return(RetCode);

memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "F1: Left, Centered");

RetCode = AFPPutTextInTableFld(hTbl, jlsfld1, szTextstring,
                               headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 1 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Begin Transaction Field 2 */
RetCode = AFPBeginField(hTbl, jlsfld2);
checkForError(RetCode,"Begin Field jlsfld2\n");

memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "F2: Left, Top");

RetCode = AFPPutTextInTableFld(hTbl, jlsfld2,
                               szTextstring, headingsFont);
checkForError(RetCode,"Put Field jlsfld2\n");

/* End Transaction Field 2 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

```

## C++ Sample

```
/* Begin Transaction Field 3 */
RetCode = AFPBeginField(hTbl, jlsfld3);
checkForError(RetCode,"Bgn Field jlsfld3\n");

memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "F3: Right, Bottom");

RetCode = AFPPutTextInTableFld(hTbl, jlsfld3,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 3 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Begin Transaction Field 4 */
RetCode = AFPBeginField(hTbl, jlsfld4);
checkForError(RetCode,"Bgn Field jlsfld4\n");

memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "F4: Left, Center");

RetCode = AFPPutTextInTableFld(hTbl, jlsfld4,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 4 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Write Transaction Field 5 */
RetCode = AFPBeginField(hTbl, jlsfld5);
checkForError(RetCode,"Begin Field jlsfld5\n");

if (RetCode)
    return(RetCode);

memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "F5: Left, Bottom");

RetCode = AFPPutTextInTableFld(hTbl, jlsfld5, szTextstring,
                               headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 5 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Begin Transaction Field 6 */
RetCode = AFPBeginField(hTbl, jlsfld6);
checkForError(RetCode,"Begin Field jlsfld6\n");

memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "F6: Right, Top");

RetCode = AFPPutTextInTableFld(hTbl, jlsfld6,
                               szTextstring, headingsFont);
checkForError(RetCode,"Put Field jlsfld6\n");

/* End Transaction Field 6 */
RetCode = AFPEndField(hTbl);
```



```

if (RetCode)
    return(RetCode);

/* Begin Transaction Field 7 */
RetCode = AFPBeginField(hTbl, jlsfld7);
checkForError(RetCode,"Bgn Field jlsfld7\n");

memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "F7: Center, Top");

RetCode = AFPPutTextInTableFld(hTbl, jlsfld7,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 7 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Begin Transaction Field 8 */
RetCode = AFPBeginField(hTbl, jlsfld8);
checkForError(RetCode,"Bgn Field jlsfld8\n");

memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "F8: Center, Top");

RetCode = AFPPutTextInTableFld(hTbl, jlsfld8,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 8 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);
/* End Transaction Row */
RetCode = AFPEndRow(hTbl, TrnsRows[0], &CurrTableDepth);
if (RetCode)
    return(RetCode);

/* Begin Transaction Row 2 */
RetCode = AFPBeginRow(hTbl, TrnsRows[1]);
checkForError(RetCode,"Bgn Row TrnRow1\n");

/* Write Transaction Field 1 */
RetCode = AFPBeginField(hTbl, jlsfld1);
checkForError(RetCode,"Bgn Row jlsfld1\n");

memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Field 1 Trans Row 2 ");

RetCode = AFPPutTextInTableFld(hTbl, jlsfld1, szTextstring,
                               headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 1 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Begin Transaction Field 2 */
RetCode = AFPBeginField(hTbl, jlsfld2);
checkForError(RetCode,"Bgn Row jlsfld2\n");

memcpy(szTextstring, NULL, sizeof(szTextstring));

```

## C++ Sample

```
strcpy(szTextstring, "Field 2 Trans Row 2 ");

RetCode = AFPPutTextInTableFld(hTbl, jlsfld2,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 2 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* Begin Transaction Field 3 */
RetCode = AFPBeginField(hTbl, jlsfld3);
checkForError(RetCode,"Bgn Row jlsfld3\n");

memcpy(szTextstring, NULL, sizeof(szTextstring));
strcpy(szTextstring, "Field 3 Trans Row 2 ");

RetCode = AFPPutTextInTableFld(hTbl, jlsfld3,
                               szTextstring, headingsFont);
if (RetCode)
    return(RetCode);

/* End Transaction Field 3 */
RetCode = AFPEndField(hTbl);
if (RetCode)
    return(RetCode);

/* End Transaction Row */
RetCode = AFPEndRow(hTbl, TrnsRows[1], &CurrTableDepth);
if (RetCode)
    return(RetCode);

return(0);
}

/*****
/* PROCESS TABLE TRANSACTIONS */
*****/
int process_table_transactions()
{
    float fValue = 0.0;

    AFPSetPos(hPage, MM_2_U1440(20), ABSOLUTE_POS,
              MM_2_U1440(80),ABSOLUTE_POS);

    /* Begin Table */
    TopThick   = .5;
    BotThick   = .5;
    LeftThick  = .5;
    RightThick = .5;

    RetCode = AFPBeginTable(hPage, MM_2_U1440(150.0),
                           IN_2_U1440(8.0), 0, MM_2_U1440(.5), MM_2_U1440(.5),
                           MM_2_U1440(.5), MM_2_U1440(.5), &hTbl);

    if (RetCode != 0) checkForError(RetCode,"Begin Table\n");

    /* Write Table Header Row */
    RetCode = write_table_hdr_row();
    if (RetCode != 0) checkForError(RetCode,"Write Header\n");
```

```

/* Write Table Trans Rows */
RetCode = write_table_trans_row();
if (RetCode !=0) checkForError(RetCode,"Write Rows\n");

/* End Table */
RetCode = AFPEndTable(&hTbl, &CurrTableDepth);
if (RetCode !=0) checkForError(RetCode,"End Table\n");

return(1);
}

/*****
/* CREATE NEW TABLE */
*****/
int create_new_table()
{
/*****
/* Define Header Fields */
*****/
RetCode = define_table_hdr_flds();
if (RetCode)
{
    checkForError(RetCode,"Define Headers/n");
    return(0);
}
/*****
/* Define Header Row */
*****/
RetCode = define_table_hdr_row();
if (RetCode)
{
    checkForError(RetCode,"Define Hdr Row/n");
    return(0);
}

/*****
/* Define Transaction Fields */
*****/
RetCode = define_table_trans_flds();
if (RetCode)
{
    checkForError(RetCode,"Define Tr Flds/n");
    return(0);
}

/*****
/* Define Transaction Row */
*****/
RetCode = define_table_trans_rows();
if (RetCode)
{
    checkForError(RetCode,"Define Tr Row/n");
    return(0);
}

/*****
/* Table Definitions Successful ! */
*****/
TableIsDefined = 1;

return(1);
}

/*****
/* CREATE NEW PAGE */
*****/

```

## C++ Sample

```
int create_page()
{
    int rc;
    long lpos;

    char buffer[15];
    char* datestr;

    /******
    /* SET GLOBAL PAGE HANDLE */
    /******
    hPage = NULL;

    /******
    /* Begin a page. */
    /******
    rc = AFPBgnPage(hOutputDoc, "Table Page", &hPage);

    checkForError(rc, "Error Creating a Page\n");

    /******
    /* Move current position (under the headings.) */
    /******
    AFPVMoveTo(hPage, IN_2_U1440(1.5));
    AFPHMoveTo(hPage, IN_2_U1440(1.5));

    return(0);
}

/******
/* END A PAGE */
/******
int end_page()
{
    int rc;
    char buffer[15];

    memset(buffer, 0x00, sizeof(buffer));

    AFPVMoveTo(hPage, IN_2_U1440(3));

    AFPHMoveTo(hPage, IN_2_U1440(7));

    sprintf(buffer, " Page %d of %d", PageNbr, NbrPages);

    /*AFPWriteString(hPage, buffer, -1, -1, LEFT_ALIGN, ' ', TRUE);*/

    rc = AFPEndPage(&hPage);

    checkForError(rc, "Error Ending a page\n");

    return(0);
}

/******
/* PROCESS PAGE INFORMATION */
/******
int process_page_information()
{
    int rc;

    /******
    /* Create a new page */
    /******
    PageNbr++;
    create_page();
}
```

```

/*****
/* Create Table Definitions */
*****/
if (!TableIsDefined)
{
    if (!create_new_table())
    {
        printf("\n create_new_table() Error!\n\n");
        return(0);
    }
}

/*****
/* Process Transactions */
*****/
if (!process_table_transactions())
{
    printf("\n process_table_transactions() Error!\n\n");
    TableProcessError = 1;
    return(0);
}

/*****
/* End a page */
*****/
end_page();

rc = AFPEndGroup(hOutputDoc);

checkForError(rc, "Error Ending a Page Group\n");

return(1);
}

/*****
/* MAIN ROUTINE */
*****/
int main()
{
    TBHANDLE toolboxSession;

    int rc, cnt;

    /*****
    /* Start Toohbox Session */
    *****/
    rc = AFPBgnSession(&toolboxSession);

    checkForError(rc, "Error Opening AFP Toolbox Session\n");

    rc = AFPBgnDoc(toolboxSession, "TBLCTST1",
        outputDocComment, TypeOfOutput, szLogicalFileName,
        AfpRecBuffer, &hOutputDoc);

    checkForError(rc, "Error Opening Output Document\n");

    /*****
    /* Define Fonts To Use */
    *****/

    printf("Defining Transaction font...\n");

    rc = AFPDefineFontByAttr(outputCodepage,
        "TIMES NEW ROMAN LATIN1", 100,
        BOLD_WT, MEDIUM_WD, NORMAL_FS,
        &transFont);

```

## C++ Sample

```
checkForError(rc, "Error Defining Transaction Font\n");

printf("Defining Heading font...\n");

rc = AFPDefineFontByAttr(outputCodepage,
                        "TIMES NEW ROMAN LATIN1", 100,
                        MEDIUM_WT, MEDIUM_WD, NORMAL_FS,
                        &headingsFont);

checkForError(rc, "Error Defining Heading Font\n");

/*****
/* Process Page/Table Information */
*****/
TableIsDefined = ProcessedData1 = ProcessedData2 = PageNbr = 0;
TableProcessError = 0;

NbrPages = 1;

/*****
/* Begin group of pages */
*****/
rc = AFPBgnGroup(hOutputDoc, "Group Name 1");
checkForError(rc, "Error Beginning a Page Group\n");

for(cnt = 0; cnt < NbrPages; cnt++)
{
    process_page_information();
}

/*****
/* End group of pages */
*****/
rc = AFPEndGroup(hOutputDoc);
checkForError(rc, "Error Ending a Page Group\n");

/*****
/* Close Output Document Data File */
*****/
rc = AFPEndDoc(&hOutputDoc);
checkForError(rc, "Error Closing Output Document\n");

rc = AFPEndSession(&toolboxSession);
checkForError(rc, "Error Closing AFP Toolbox Session\n");

if (!TableProcessError)
    printf("\nTables Sample Test TBLSUBR1.C Completed Successfully!\n");

return 0;
}
```

---

## Sample in COBOL

```
*****
*   COBOL PROGRAM -- AFP Toolbox Table Test                               *
*-----*
*
*   This testcase will construct a three row table                       *
*   beginning 80 mm from the top of the page.                           *
*   The header fields are left, center, and right aligned.              *
*   The transaction rows have 8 fields with various text                 *
*   formatting options as described in the printed output.              *
*
*****

PROGRAM-ID. "TBLCBLT2".
ENVIRONMENT DIVISION.
```

CONFIGURATION SECTION.  
 SOURCE-COMPUTER. IBM.  
 OBJECT-COMPUTER. IBM.

INPUT-OUTPUT SECTION.  
 FILE-CONTROL.

```
*****
*
*                               DATA DIVISION
*
*                               *****
```

DATA DIVISION.  
 FILE SECTION.

WORKING-STORAGE SECTION.

```
01 FONT-IDS.
   05 TIM10MED          PIC 9(4) BINARY VALUE 0.
   05 TIM10BOLD         PIC 9(4) BINARY VALUE 0.
01 FIELD-IDS.
   05 FIELDH1           PIC 9(8) BINARY.
   05 FIELDH2           PIC 9(8) BINARY.
   05 FIELDH3           PIC 9(8) BINARY.
   05 FIELDT1           PIC 9(8) BINARY.
   05 FIELDT2           PIC 9(8) BINARY.
   05 FIELDT3           PIC 9(8) BINARY.
   05 FIELDT4           PIC 9(8) BINARY.
   05 FIELDT5           PIC 9(8) BINARY.
   05 FIELDT6           PIC 9(8) BINARY.
   05 FIELDT7           PIC 9(8) BINARY.
   05 FIELDT8           PIC 9(8) BINARY.
01 ROW-IDS.
   05 ROW1              PIC 9(3) BINARY VALUE 0.
   05 ROW2              PIC 9(3) BINARY VALUE 0.
```

COPY ATXCBVAR.

```
*****
*
*                               PROCEDURE DIVISION
*
*                               *****
```

PROCEDURE DIVISION.

```
*****
*
* MAINLINE
*
*       The mainline logic is as follows:
*
*       Initialize session & Define Objects;
*       Begin a document;
*       Begin a page;
*       Build a table;
*       End the page;
*       End the document & session;
*
* *****
```

MAINLINE.

```
      MOVE MM TO AFP-UNIT-OF-MEASURE.
      PERFORM AFPSUNI.
```

## COBOL Sample

PERFORM SETUP-TBOX.

MOVE "Page 1" TO AFP-PAGE-NAME.  
PERFORM AFPBPAGE.  
SET AFP-CURRENT-HANDLE TO AFP-PAGE-HANDLE.

MOVE TIM10BOLD TO AFP-FONT-ID.  
PERFORM AFPSFONT.

MOVE 30 TO AFP-X-COORDINATE.  
MOVE ABSOLUTE-POS TO AFP-X-REF-COORD-SYS.  
MOVE 80 TO AFP-Y-COORDINATE.  
MOVE ABSOLUTE-POS TO AFP-Y-REF-COORD-SYS.  
PERFORM AFPSPOS.

\*-----\*  
\* Begin the table. \*  
\*-----\*

MOVE 150.0 TO AFP-TABLE-WIDTH.  
MOVE 180.0 TO AFP-MAX-TABLE-DEPTH.  
MOVE ORIENT0 TO AFP-TABLE-ROTATION.  
MOVE 0.5 TO AFP-TOP-THICKNESS.  
MOVE 0.5 TO AFP-BOTTOM-THICKNESS.  
MOVE 0.5 TO AFP-LEFT-THICKNESS.  
MOVE 0.5 TO AFP-RIGHT-THICKNESS.  
PERFORM AFPBTABL.

\*-----\*  
\* \* \*  
\* Write the header row in the table. \*  
\* \* \*  
\*-----\*

MOVE ROW1 TO AFP-ROW-ID.  
PERFORM AFPBROW.

\*-----\*  
\* Write the first field in the header \*  
\*-----\*

MOVE FIELDH1 TO AFP-FIELD-ID.  
PERFORM AFPBFLD.  
MOVE TIM10BOLD TO AFP-FONT-ID.  
PERFORM AFPSFONT.

MOVE 14 TO AFP-STRING-LENGTH.  
MOVE "Left/Vercenter" TO AFP-CHARACTER-STRING.  
PERFORM AFPPCHS.

PERFORM AFPEFLD.

\*-----\*  
\* Write the second field in the header \*  
\*-----\*

MOVE FIELDH2 TO AFP-FIELD-ID.  
PERFORM AFPBFLD.

MOVE 18 TO AFP-STRING-LENGTH.  
MOVE "Centered/Vercenter" TO AFP-CHARACTER-STRING.  
PERFORM AFPPCHS.

PERFORM AFPEFLD.

\*-----\*  
\* Write the third field in the header \*  
\*-----\*



```

*-----*
    MOVE FIELDH3 TO AFP-FIELD-ID.
    PERFORM AFPBFLD.

    MOVE 15 TO AFP-STRING-LENGTH.
    MOVE "Right/Vercenter" TO AFP-CHARACTER-STRING.
    PERFORM AFPPCHS.

    PERFORM AFPEFLD.
    PERFORM AFPEROW.
*-----*
*                                           *
* Write the first transaction row in the table. *
*                                           *
*-----*

    MOVE ROW2 TO AFP-ROW-ID.
    PERFORM AFPBROW.

    MOVE TIM10MED TO AFP-FONT-ID.
    PERFORM AFPSFONT.

    MOVE FIELDT1 TO AFP-FIELD-ID.
    PERFORM AFPBFLD.
    MOVE "F1: Left, Centered" TO AFP-CHARACTER-STRING
    MOVE 18 TO AFP-STRING-LENGTH.
    PERFORM AFPPCHS.
    PERFORM AFPEFLD.

    MOVE FIELDT2 TO AFP-FIELD-ID.
    PERFORM AFPBFLD.
    MOVE "F2: Left, Top" TO AFP-CHARACTER-STRING.
    MOVE 13 TO AFP-STRING-LENGTH.
    PERFORM AFPPCHS.
    PERFORM AFPEFLD.

    MOVE FIELDT3 TO AFP-FIELD-ID.
    MOVE FIELDT3 TO AFP-FIELD-ID.
    PERFORM AFPBFLD.
    MOVE "F3: Right, Bottom" TO AFP-CHARACTER-STRING.
    MOVE 17 TO AFP-STRING-LENGTH.
    PERFORM AFPPCHS.
    PERFORM AFPEFLD.

    MOVE FIELDT4 TO AFP-FIELD-ID.
    PERFORM AFPBFLD.
    MOVE "F4: Left, Centered" TO AFP-CHARACTER-STRING.
    MOVE 18 TO AFP-STRING-LENGTH.
    PERFORM AFPPCHS.
    PERFORM AFPEFLD.

    MOVE FIELDT5 TO AFP-FIELD-ID.
    PERFORM AFPBFLD.
    MOVE "F5: Center, Bot" TO AFP-CHARACTER-STRING.
    MOVE 15 TO AFP-STRING-LENGTH.
    PERFORM AFPPCHS.
    PERFORM AFPEFLD.

    MOVE FIELDT6 TO AFP-FIELD-ID.
    PERFORM AFPBFLD.
    MOVE "F6: Right, Top" TO AFP-CHARACTER-STRING.
    MOVE 14 TO AFP-STRING-LENGTH.
    PERFORM AFPPCHS.
    PERFORM AFPEFLD.

    MOVE FIELDT7 TO AFP-FIELD-ID.

```

## COBOL Sample

```
PERFORM AFPBFLD.  
  MOVE "F7: Center, Bottom" TO AFP-CHARACTER-STRING.  
  MOVE 18 TO AFP-STRING-LENGTH.  
PERFORM AFPPCHS.  
PERFORM AFPEFLD.
```

```
  MOVE FIELDT8 TO AFP-FIELD-ID.  
PERFORM AFPBFLD.  
  MOVE "F8: Center, Top" TO AFP-CHARACTER-STRING.  
  MOVE 17 TO AFP-STRING-LENGTH.  
PERFORM AFPPCHS.  
PERFORM AFPEFLD.
```

```
PERFORM AFPEROW.
```

```
*-----*  
*                                           *  
*   Write the second transaction row (with some empty fields) *  
*                                           *  
*-----*
```

```
  MOVE ROW2 TO AFP-ROW-ID.  
PERFORM AFPBROW.
```

```
  MOVE TIM10MED TO AFP-FONT-ID.  
PERFORM AFPSFONT.
```

```
  MOVE FIELDT1 TO AFP-FIELD-ID.  
PERFORM AFPBFLD.  
  MOVE "Field 1 Trans Row 2" TO AFP-CHARACTER-STRING.  
  MOVE 19 TO AFP-STRING-LENGTH.  
PERFORM AFPPCHS.  
PERFORM AFPEFLD.
```

```
  MOVE FIELDT2 TO AFP-FIELD-ID.  
PERFORM AFPBFLD.  
  MOVE "Field 2 Trans Row 2" TO AFP-CHARACTER-STRING.  
  MOVE 19 TO AFP-STRING-LENGTH.  
PERFORM AFPPCHS.  
PERFORM AFPEFLD.
```

```
  MOVE FIELDT3 TO AFP-FIELD-ID.  
PERFORM AFPBFLD.  
  MOVE "Field 3 Trans Row 3" TO AFP-CHARACTER-STRING.  
  MOVE 19 TO AFP-STRING-LENGTH.  
PERFORM AFPPCHS.  
PERFORM AFPEFLD.  
PERFORM AFPEROW.
```

```
PERFORM AFPETABL.  
PERFORM AFPEPAGE.
```

```
PERFORM END-PROCESSING.
```

```
  DISPLAY "TBLCBLT2 COMPLETED".  
  STOP RUN.
```

```
*-----*  
*                                           *  
*   SETUP-TBOX.                               *  
*                                           *  
*       Initialize the AFP API session;        *  
*       Set the output characteristics;        *  
*       Begin a document;                     *  
*       Define the fonts.                     *  
*       Define the fields;                     *  
*                                           *
```

```

*           Define the rows;                               *
*                                                         *
*-----*
SETUP-TBOX.

PERFORM AFPINIT.

*-----*
*                                                         *
* Set up the document characteristics.                     *
*                                                         *
*-----*

MOVE "FILEOUT" TO AFP-OUTPUT-FILENAME.

MOVE FILED TO AFP-OUTPUT-TYPE.
PERFORM AFPBDOC.
SET AFP-CURRENT-HANDLE TO AFP-DOCUMENT-HANDLE.
MOVE MM TO AFP-UNIT-OF-MEASURE.

*-----*
*                                                         *
* Define the fonts.                                       *
*                                                         *
*-----*

MOVE "T1V10500" TO AFP-CODE-PAGE.
MOVE "TIMES NEW ROMAN LATIN1" TO AFP-DESCRIPTIVE-NAME.
MOVE 100 TO AFP-POINT-SIZE.
MOVE MEDIUM TO AFP-WEIGHT.
MOVE NORML TO AFP-FONT-WIDTH.
MOVE ROMAN TO AFP-STYLE.
PERFORM AFPDFNTAT.
MOVE AFP-FONT-ID TO TIM10MED.

MOVE "T1V10500" TO AFP-CODE-PAGE.
MOVE "TIMES NEW ROMAN LATIN1" TO AFP-DESCRIPTIVE-NAME.
MOVE 100 TO AFP-POINT-SIZE.
MOVE BOLD TO AFP-WEIGHT.
MOVE NORML TO AFP-FONT-WIDTH.
MOVE ROMAN TO AFP-STYLE.
PERFORM AFPDFNTAT.
MOVE AFP-FONT-ID TO TIM10BOLD.

*-----*
* Define the header fields                                *
*-----*

MOVE FOLEFT TO AFP-FORMAT-OPTION.
MOVE 0 TO AFP-ALIGNMENT-POSITION.
MOVE VERCENTER TO AFP-VERTICAL-FORMAT.
MOVE 0.0 TO AFP-LEFT-MARGIN.
MOVE 0.0 TO AFP-RIGHT-MARGIN.
MOVE DEFAULT-LSP TO AFP-LINE-SPACING.
MOVE TXTOR0-0 TO AFP-TEXT-ORIENTATION.
MOVE 0 TO AFP-SHADE-PERCENT.
PERFORM AFPDFLD.
MOVE AFP-FIELD-ID TO FIELDH1.

MOVE FOCENTER TO AFP-FORMAT-OPTION.
PERFORM AFPDFLD.
MOVE AFP-FIELD-ID TO FIELDH2.

MOVE FORIGHT TO AFP-FORMAT-OPTION.
PERFORM AFPDFLD.

```

## COBOL Sample

MOVE AFP-FIELD-ID TO FIELDH3.

```
*-----*
*   Define the transaction fields                                   *
*-----*
```

MOVE FOLEFT TO AFP-FORMAT-OPTION.  
MOVE 0 TO AFP-ALIGNMENT-POSITION.  
MOVE VERCENTER TO AFP-VERTICAL-FORMAT.  
MOVE 0.0 TO AFP-LEFT-MARGIN.  
MOVE 0.0 TO AFP-RIGHT-MARGIN.  
MOVE DEFAULT-LSP TO AFP-LINE-SPACING.  
MOVE TXTOR0-0 TO AFP-TEXT-ORIENTATION.  
MOVE 0 TO AFP-SHADE-PERCENT.  
PERFORM AFPDFLD.  
MOVE AFP-FIELD-ID TO FIELDT1.

MOVE FOLEFT TO AFP-FORMAT-OPTION.  
MOVE VERTOP TO AFP-VERTICAL-FORMAT.  
PERFORM AFPDFLD.  
MOVE AFP-FIELD-ID TO FIELDT2.

MOVE FORIGHT TO AFP-FORMAT-OPTION.  
MOVE VERBOTTOM TO AFP-VERTICAL-FORMAT.  
PERFORM AFPDFLD.

MOVE AFP-FIELD-ID TO FIELDT3.  
MOVE FOLEFT TO AFP-FORMAT-OPTION.  
MOVE VERCENTER TO AFP-VERTICAL-FORMAT.  
PERFORM AFPDFLD.  
MOVE AFP-FIELD-ID TO FIELDT4.

MOVE FOLEFT TO AFP-FORMAT-OPTION.  
MOVE VERBOTTOM TO AFP-VERTICAL-FORMAT.  
PERFORM AFPDFLD.  
MOVE AFP-FIELD-ID TO FIELDT5.

MOVE FORIGHT TO AFP-FORMAT-OPTION.  
MOVE VERTOP TO AFP-VERTICAL-FORMAT.  
PERFORM AFPDFLD.  
MOVE AFP-FIELD-ID TO FIELDT6.

MOVE FOCENTER TO AFP-FORMAT-OPTION.  
MOVE VERBOTTOM TO AFP-VERTICAL-FORMAT.  
PERFORM AFPDFLD.  
MOVE AFP-FIELD-ID TO FIELDT7.

MOVE FOCENTER TO AFP-FORMAT-OPTION.  
MOVE VERTOP TO AFP-VERTICAL-FORMAT.  
PERFORM AFPDFLD.  
MOVE AFP-FIELD-ID TO FIELDT8.

```
*-----*
*   Define the header row                                         *
*-----*
```

MOVE 3 TO AFP-NUMBER-COLUMNS.  
MOVE 1 TO AFP-NUMBER-SUBROWS.  
MOVE 15 TO AFP-SUBROW-DEPTH(1).  
MOVE FIELDH1 TO AFP-COLUMN-ARRANGE (1).  
MOVE 40.0 TO AFP-COLUMN-WIDTH (1).  
MOVE FIELDH2 TO AFP-COLUMN-ARRANGE (2).  
MOVE 80.0 TO AFP-COLUMN-WIDTH (2).  
MOVE FIELDH3 TO AFP-COLUMN-ARRANGE (3).  
MOVE 30.0 TO AFP-COLUMN-WIDTH (3).  
PERFORM AFPDROW.  
MOVE AFP-ROW-ID TO ROW1.

```

*-----*
* Define the transaction row.                                *
*-----*

    MOVE 3 TO AFP-NUMBER-COLUMNS.
    MOVE 4 TO AFP-NUMBER-SUBROWS.
    MOVE 25 TO AFP-SUBROW-DEPTH(1).
    MOVE 15 TO AFP-SUBROW-DEPTH(2).
    MOVE 20 TO AFP-SUBROW-DEPTH(3).
    MOVE 30 TO AFP-SUBROW-DEPTH(4).
    MOVE 45.0 TO AFP-COLUMN-WIDTH (1).
    MOVE 70.0 TO AFP-COLUMN-WIDTH (2).
    MOVE 35.0 TO AFP-COLUMN-WIDTH (3).
    MOVE FIELDT1 TO AFP-COLUMN-ARRANGE (1).
    MOVE FIELDT2 TO AFP-COLUMN-ARRANGE (2).
    MOVE FIELDT3 TO AFP-COLUMN-ARRANGE (3).
    MOVE FIELDT1 TO AFP-COLUMN-ARRANGE (4).
    MOVE FIELDT4 TO AFP-COLUMN-ARRANGE (5).
    MOVE FIELDT3 TO AFP-COLUMN-ARRANGE (6).
    MOVE FIELDT5 TO AFP-COLUMN-ARRANGE (7).
    MOVE FIELDT4 TO AFP-COLUMN-ARRANGE (8).
    MOVE FIELDT6 TO AFP-COLUMN-ARRANGE (9).
    MOVE FIELDT5 TO AFP-COLUMN-ARRANGE (10).
    MOVE FIELDT7 TO AFP-COLUMN-ARRANGE (11).
    MOVE FIELDT8 TO AFP-COLUMN-ARRANGE (12).
    PERFORM AFPDROW.
    MOVE AFP-ROW-ID TO ROW2.

*-----*
*
* Terminate the AFP API session and the program.            *
*
*-----*

END-PROCESSING.
    PERFORM AFPEDOC.
    PERFORM AFPEND.

    COPY ATXCBPRF.
    COPY ATXTRIM.

END PROGRAM "TBLCBLT2".

```

## COBOL Sample

---

## Notices

## Notices

This information was developed for products and services offered in the USA.

IBM may not offer the products, services or features discussed in this document in your country. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program or service is not intended to state or imply that only that IBM product, program or service may be used. Any functionally equivalent product, program or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10594-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer or express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product and/or the program described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the material for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one), and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
Mail Drop 001W  
Boulder, CO 80301  
U.S.A.



Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

## COPYRIGHT LICENSE

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

## Programming Interfaces

This publication is intended to help the customer program AFP applications with high-level programming languages, such as C, C++, COBOL, and RPG. This publication documents General-Use Programming Interface and Associated Guidance Information provided by AFP Toolbox.

General-Use Programming Interfaces allow the customer to write programs that obtain the services of AFP Toolbox.

## Trademarks

The following terms appear in this publication and are either trademarks or registered trademarks of the IBM Corporation:

AFP	MVS
Advanced Function Presentation	MVS/ESA
AIX	OS/2
AS/400	OS/390
BookManager	OS/400
CICS	Print Services Facility
CICS/ESA	PTX
GDDM	RISC System/6000
IBM	RS/6000
Infoprint	SP
IPDS	System/370
Language Environment	

The following trademarks are licensed to other companies as indicated:

**UNIX** UNIX is a registered trademark of The Open Group.

### Windows

Windows and Windows NT are registered trademarks of Microsoft® Corporation.

## **Notices**

### **EuroReady**

The AFP Toolbox for Multiple Operating Systems (Toolbox) is capable of processing data containing the euro sign. Font character sets and code pages that contain and map the euro sign consistently with the application must be present either in a host library or in the printer. AFP fonts that support the euro sign are included in the AFP Font Collection.

### **Year 2000 Ready**

Toolbox does not have date dependencies and is therefore Year 2000 ready. When used in accordance with its associated documentation, Toolbox is capable of correctly processing, providing, and receiving date data within and between the twentieth and twenty-first centuries, provided all other products used with Toolbox (including software, hardware and firmware) properly exchange accurate date data with it.

---

# Glossary

---

## Source Identifiers

This publication includes terms and definitions from:

- The *American National Dictionary for Information Processing Systems*, copyright 1982 by the Computer and Business Equipment Manufacturers Association (CBEMA). Copies can be purchased from the

American National Standards Institute,  
1430 Broadway  
New York, New York 10018

Definitions are identified by the symbol (A) after the definition.

- The *Information Technology Vocabulary*, developed by the Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Committee (ISO/IEC JTC1/SC1).

Definitions of published segments of the vocabularies are identified by the symbol (I) after the definition; definitions from draft international standards, draft proposals, and working papers in development by the ISO/IEC JTC1/SC1 vocabulary subcommittee are identified by the symbol (T) after the definition, indicating final agreement has not yet been reached among participating members.

---

## References

This glossary uses the following cross-references:

### Deprecated term for

Indicates that the term should not be used (because it is obsolete, misleading, ambiguous, or jargonistic) and refers to the preferred term. For a deprecated term, the commentary contains only this reference; the deprecated term is not defined.

### Synonymous with

Appears in the commentary of a preferred term and identifies less desirable or less specific terms that have the same meaning. The commentaries of the less desirable or less specific terms refer back to the preferred term with the *Synonym for* reference words.

### Synonym for

Appears in the commentary of a less desirable or less specific term and identifies the preferred term that has the same meaning.

### Contrast with

Refers to a term that has an opposite or substantively different meaning.

### See

Refers to a multiple-word term in which this term appears.

### See also

Refers to related terms that have similar (but not synonymous) meanings.

## A

**abend.** An abnormal end of task before the task's completion because of an error condition.

**absolute positioning.** The establishment of a position within a coordinate system as an offset from the coordinate system origin. Contrast with *relative positioning*.

**addressable position.** A position in a presentation space or on a physical medium that can be identified by a coordinate from the coordinate system of the presentation space or physical medium. See also *picture element*. Synonymous with *position*.

**addressable point.** For page printers, any point in a presentation surface that can be addressed. Synonymous with *picture element*.

**Advanced Function Presentation (AFP).** A set of licensed programs that use the all-points-addressable concept to view and print text and graphics.

**Advanced Function Printing data stream (AFPDS).** The data stream supported by IBM's Advanced Function Presentation products.

**AFP.** (1) Advanced Function Presentation. (2) Advanced Function Printing.

**AFPDS.** Advanced Function Printing data stream.

**AFP resource.** An object that contains printing instructions and sometimes data that can be used in a print job. It is a special kind of AFP object that can be stored in a library and can be shared by different printer files and different pages within a printer file. Coded fonts, code pages, font character sets, form definitions, overlays, page definitions, and page segments are AFP resources.

## Glossary

**all points addressable (APA).** (1) The ability to address, reference, and position text, overlays, and images at any defined position or pel on the printable area of the paper. This capability depends on the ability of the hardware to address and to display each picture element. (2) In computer graphics, pertaining to the ability to address and display or not display each picture element (pel) on a display surface. (3) See also *picture element*.

**American National Standard Code for Information Interchange (ASCII).** The standard code, using a coded character set consisting of 7-bit coded characters (8-bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

**APA.** *All-points addressable.*

**application program.** (1) A program that performs a particular data processing task, such as inventory control or payroll. (2) A program that produces the print file.

**area.** An AFP Toolbox element that can be defined and used repeatedly in a document or page.

**ASCII.** American National Standard Code for Information Interchange.

**attribute.** A property or characteristic of one or more entities. (T)

## B

**background.** (1) The part of a presentation space that is not occupied with object data. (2) In GOCA, that portion of a drawing primitive that is mixed into the presentation space under the control of the current values of the background mix and background color attributes. (3) In GOCA, that portion of a character cell that does not represent a character. (4) The space of a bar code symbol.

**bar code.** A code representing characters by sets of parallel bars of varying thickness and separation that are read optically by transverse scanning. (I)

**bar code object.** The object containing the structured fields required to present bar code information on a page, a page segment, or an overlay.

**baseline.** The imaginary line on which successive characters are aligned in the inline direction.

**baseline axis.** The axis along which successive lines of text are placed.

**baseline direction.** The direction in which successive lines of text appear on a logical page.

**baseline increment.** The distance between successive sequential baselines.

**bin.** A paper supply on cut-sheet printers. See also *cassette*.

**boilerplate.** (1) A frequently used segment of stored text that can be combined with other text to create a new document. (T) (2) In word processing and desktop publishing, text that is stored for repeated use in various documents; for example, the wording of an edition notice.

## C

**carriage control character.** An optional character in an input data record that specifies a write, space, or skip operation.

**cassette.** In a cut-sheet printer, a movable paper storage enclosure. See also *bin*.

**channel code.** A number from 1 to 12 that identifies a position in the forms control buffer or a page definition. A carriage control character can select a position defined by a particular channel code.

**CHAR.** A data type for architecture syntax, indicating one or more bytes to be interpreted as character information.

**character.** (1) A member of a set of elements that is used for the representation, organization, or control of data. Characters can be letters, digits, punctuation marks, or other symbols. (T) (2) A character is often represented in the form of a spatial arrangement of adjacent or connected strokes or in the form of other physical conditions in data media. (3) A letter, numeral, punctuation mark, or special graphic used for the production of text. (4) A byte of data. (5) See also *graphic character*.

**character baseline.** A reference line within a character box on which the character reference point lies. The character baseline is used to orient and position the initial point of a character.

**character data.** Data in the form of letters and special characters, such as punctuation marks. See also *numeric data*.

**character increment.** The distance between the current print position and the next print position.

**character metrics.** Measurement information that defines individual character values such as height, width, and space. Character metrics can be expressed in specific fixed units, such as pels, or in relative units that are independent of both the resolution and size of the font. Character metrics are often included as part of the general term font metrics. See also *font metrics*.

**character rotation.** The alignment of a character with respect to its character baseline, measured in degrees in a clockwise direction. Examples are 0°, 90°, 180°, and 270°. 0° character rotation exists when a character is in its customary alignment with the baseline.

**character set.** (1) A collection of characters that is composed of some descriptive information and the character shapes themselves. (2) A group of characters used for a specific reason, for example, the set of characters a keyboard contains. (3) Synonym for *font character set*. (4) See also *coded font*.

**character string.** A sequence of characters.

**CICS/ESA®.** Customer Information Control System/Enterprise System Architecture.

**CODE.** A data type for architecture syntax, indicating an architected constant to be interpreted as defined by the architecture.

**coded font.** An AFP font that associates a code page and a font character set.

**code page.** Part of an AFP font that associates code points and character identifiers. A code page also identifies undefined code points. See also *coded font*.

**code point.** A 1-byte code representing one of 256 potential characters.

**composed text.** Deprecated term for *presentation text*.

**compound document.** A collection of data objects that makes up the document's content, the associated resources, and formatting specifications for the content.

**compression algorithm.** An algorithm used to compress image data. Compression of image data can decrease the volume of data required to represent an image.

**continuous forms.** A series of connected forms that feed continuously through a printing device. The connection between the forms is perforated, enabling the user to tear them apart. Before printing, the forms are folded in a stack arrangement with the folds along the perforations. Contrast with *cut-sheet paper*.

**control character.** A character that starts, changes, or stops any operation that affects recording, processing, transmitting, or interpreting data (such as carriage return, font change, and end of transmission).

**coordinate system.** A Cartesian coordinate system. An example is the image coordinate system that uses the fourth quadrant with positive values for the Y-axis. The origin is the upper left-hand corner of the fourth quadrant. A pair of values in the X and Y axes corresponds to one image point that corresponds to a single image data element.

**coordinates.** A pair of values that specify a position in a coordinate space.

**copies.** See *copy group*.

**copy files.** Files shipped with AFP Toolbox to aid in developing user programs. The files contain such items as AFP Toolbox variables, return codes, constants for variables, paragraphs that invoke AFP Toolbox procedures, and other programs or subprograms.

**copy group.** A subset of a form definition that lets different modifications be made to multiple copies of the input data. Modifications can include suppression of some data fields from printing as well as the use of different overlays.

**core interchange font.** See *IBM Core Interchange font*.

**current position.** The position identified by the current presentation space coordinates. For example, the coordinate position reached after the execution of a drawing order. See also *current baseline presentation coordinate* and *current inline presentation coordinate*. Contrast with *given position*.

**Customer Information Control System (CICS).** An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining databases.

**cut-sheet paper.** The medium that is cut into uniform-size sheets before it is loaded into the printer. Contrast with *continuous forms*.

## D

**data control block (DCB).** A control block used by access method routines in storing and retrieving data.

**data map.** An internal object in a page definition that specifies fonts, page segments, fixed text, page size, and the placement and orientation of text. Synonymous with *page format*.

**data object.** An AFP data object contains a single type of presentation data; that is, either presentation text, vector graphics, raster images, or bar codes, plus all the controls required to present the data.

**data stream.** A continuous stream of data that has a defined format. An example of a defined format is a structured field.

**DCB.** Data control block.

**DCF.** Document Composition Facility.

**default.** Pertaining to an attribute, value, or option that is assumed when none is explicitly specified. (I)

## Glossary

**direction.** In GOCA, an attribute controlling the direction in which a character string grows relative to the inline direction. Values are: left-to-right, right-to-left, top-to-bottom, and bottom-to-top.

**disabled.** A condition of the printer (physically selected) in which the printer is not available to the host processor. Contrast with *enabled*.

**document.** A file containing an AFP data stream document. An AFP data stream document is bounded by Begin Document and End Document structured fields and can be created using a text formatter such as AFP Toolbox.

**document fidelity.** Synonym for *fidelity*.

**Document Composition Facility (DCF).** An IBM licensed program that provides a text formatter called SCRIPT/VS. SCRIPT/VS can process files marked up with a unique set of controls and tags.

**duplex printing.** Printing on both sides of a sheet of paper. Contrast with *simplex printing*. See also *normal duplex printing* and *tumble duplex printing*.

## E

**EBCDIC.** Extended binary-coded decimal interchange code.

**electronic forms.** A collection of constant data that is electronically composed in the host processor and that can be merged with variable data on a page during printing.

**electronic overlay.** A collection of constant data, such as lines, shading, text, boxes, or logos, that is electronically composed in the host processor and stored in a library, and that can be merged with variable data during printing. Contrast with *page segment*. See also *page overlay* and *medium overlay*.

**element.** (1) A bar or space in a bar code character or a bar code symbol. (2) A structured field in a document-content architecture data stream.

**enabled.** (1) Pertaining to a state of the processing unit that allows the occurrence of certain types of interruptions. (2) A condition of the printer (physically selected) in which the printer is available to the host processor for normal work. (3) Contrast with *disabled*.

**exception.** A condition that exists when the printer:

- Detects an invalid or unsupported command, order, control, or parameter value from the host
- Finds a condition requiring host-system notification
- Detects a condition that requires the host system to resend data

**extended binary-coded decimal interchange code (EBCDIC).** A coded character set consisting of eight-bit coded characters.

## F

**FCB.** Forms control buffer.

**fidelity.** The ability to replicate faithfully the appearance of document content on presentation surfaces. Synonymous with *document fidelity*.

**field.** The area where you put text in a table.

**FLIP.** Font Library Indexing Program.

**font.** (1) A family or assortment of characters of a given size and style; for example, 9 point Bodoni Modern. (A) (2) One size and one typeface in a particular type family, including letters, numerals, punctuation marks, special characters, and ligatures.

**font character set.** Part of an AFP font that contains the raster patterns, identifiers, and descriptions of characters. Synonymous with *character set*. See also *coded font*.

**font metrics.** Measurement information that defines individual character values such as height, width, and space, as well as overall font values, such as averages and maximums. Font metrics can be expressed in specific fixed units, such as pels, or in relative units that are independent of both the resolution and size of the font. See also *character metrics*.

**font object.** A resource object that contains the description of a font.

**Font Object Content Architecture (FOCA).** An architected collection of constructs used to describe fonts and to interchange those font descriptions.

**font width.** The average character width expressed in 1440th of an inch. For proportionally spaced fonts, the font width is 1/3 of the point size converted to 1440th of an inch (or the width of the average character escapement box). For fixed pitch fonts, the font width is calculated by dividing 1440 by the pitch. For mixed pitch fonts, the font width is the width of the space character (usually 120). See also *pitch*.

**form.** The paper on which output data is printed by a printer. Synonymous with *physical page*, *medium*, and *sheet*.

**format.** (1) The shape, size, and general makeup of a printed document. (2) To prepare a document for printing. (3) The arrangement of text on the page.

**formatter.** A computer program that prepares a source document for printing.

**form definition.** A resource used by PSF that defines the characteristics of the form that includes overlays to be used (if any), text suppression, the paper source (for cut-sheet printers), the position of page data on the form, and the number and modifications of a page.



**forms control buffer (FCB).** A buffer for controlling the vertical format of printed output. The forms control buffer is a line-printer control that is similar to the punched-paper, carriage-control tape used on IBM 1403 printers.

## G

**GDDM®.** Graphical Data Display Manager.

**GMS.** graphics model space.

**GOCA.** Graphic Object Content Architecture.

**GPS.** graphics presentation space.

**graphic character.** A visual representation of a character, other than a control character, that is normally produced by writing, printing, or displaying. (T)

**Graphical Data Display Manager.** An IBM licensed program containing utilities for creating, saving, editing, and displaying visual data such as page segments, charts, images, vector graphics, composites (of text, graphics, and images), and scanned data.

**graphics data.** Data containing lines, arcs, markers, and other constructs that describe a picture.

**graphics data.** Data containing lines, arcs, markers, and other constructs that describe a picture.

**graphics model space (GMS).** A two-dimensional conceptual space in which a picture is constructed. All model transforms are completed before a picture is constructed in a graphics model space. Contrast with *graphics presentation space*. Synonymous with *model space*.

**graphics presentation space (GPS).** A two-dimensional conceptual space in which a picture is constructed. In this space, graphics drawing orders are defined. The picture can then be mapped onto an output medium. All viewing transforms are completed before the picture is generated for presentation on an output medium. An example of a graphics presentation space is the abstract space containing graphics pictures defined in an IPDS Write Graphics Control command. Contrast with *graphics model space*.

**group.** An object in a file used to define a named, logical grouping of sequential pages.

## H

**handle.** An ID that identifies the session and the state of the document in which AFP Toolbox is operating. AFP Toolbox uses handles to keep track of which state in the document it is processing at any given time.

**hanging indent.** When the first line of a paragraph begins at the left margin and all subsequent lines are indented from the left margin by the specified amount.

**hexadecimal.** Pertaining to a numbering system with base of 16; valid numbers use the digits 0 through 9 and characters A through F, where A represents 10 and F represents 15.

## I

**IBM Core Interchange font.** (1) A uniformly spaced, typographic font with specialized characters for different languages. (2) A group of fonts supplied with Print Services Facility Version 2. These fonts include the Courier, Helvetica, and Times New Roman type families. Using the IBM Core Interchange fonts increases the fidelity of documents exchanged between different systems and applications.

**image.** An electronic representation of a picture. An image can also be generated directly by software without reference to an existing picture.

**image data.** A pattern of bits with 0 and 1 values that define the pels in an image. (A 1-bit is a toned pel.)

**image object.** An object that contains image data.

**Image Object Content Architecture (IOCA).** An architected collection of constructs used to interchange and present images.

**image segment.** A set of image data parameters and image data that appear between begin and end segment self-defining fields.

**IM image.** One of the two types of images used in AFP data streams. An IM image is device- and resolution-dependent, bi-level, uses pel-to-pel mapping for presentation, and cannot be compressed or scaled. Contrast with *IO image*.

**indexing tag.** A structured field containing an attribute name and value that lets you identify information in a document, such as a page.

**inline.** (1) The direction of successive characters in a line of text. Synonymous with *inline direction*. (2) A resource contained within the print file so that a reference to a resource library is not needed.

**inline axis.** The axis along which successive characters in a line are placed.

**inline direction.** (1) The direction in which successive characters appear in a line of text. (2) In GOCA, the direction specified by the character angle attribute.

**inline margin.** The inline coordinate that identifies the initial addressable position for a line of text.

## Glossary

**interface.** A shared boundary. An interface can be a hardware component used to link two devices, or it can be a portion of storage or registers accessed by two or more computer programs. (A)

**IO image.** An image object containing IOCA constructs.

**IOCA.** Image Object Content Architecture.

## J

**JAN.** Japanese Article Numbering. A type of bar code.

## L

**landscape.** Pertaining to a display or hardcopy of a page with greater width than height. Contrast with *portrait*.

**library.** A data file that contains files and control information that lets each file be accessed individually.

**line data.** Data prepared for printing on a line printer, such as a 3800 Model 1.

**line printer.** A device that prints a line of characters as a unit. (I) (A) Contrast with *page printer*.

**local identifier.** An identifier that is mapped by the environment to a named resource.

**location.** A site within a data stream. A location is specified in terms of an offset in the number of structured fields from the beginning of a data stream, or in the number of bytes from another location within the data stream.

**logical page.** A presentation space. One or more object areas or data blocks can be mapped to a logical page. A logical page has specifiable characteristics. Examples of specifiable page characteristics are size, shape, orientation, and offset. The shape of a page is the shape of a rectangle. Orientation and offset are specified relative to a medium coordinate system. Contrast with *physical page*.

**logical page offset.** The distance between the medium origin and the logical page origin.

**logical page origin.** The point on the logical page that represents  $X_p=0$ ,  $Y_p=0$  in the  $X_p$ ,  $Y_p$  coordinate system.

## M

**magnetic ink character recognition (MICR).** Recognition of characters printed with ink that contains particles of a magnetic material. (I) (A)

**MBCS.** See *Multibyte Character Set*(I) (A)

**media.** Plural of medium. See also *medium*.

**medium.** A base coordinate space from which all other coordinate spaces either directly or indirectly originate. A medium is mapped to a presentation surface in a device-dependent manner. Synonymous with *form*, *physical page*, and *sheet*.

**medium map.** An internal object in a form definition that controls the modifications to a form, page placement, and overlays. Synonymous with *copy group*.

**media origin.** A page is placed on a medium, or form, relative to the media origin. The media origin is located at the very top-left corner of the form.

**medium overlay.** An electronic overlay that is invoked by the medium map of a form definition for printing at a fixed position on the form. See also *page overlay* and *overlay*.

**MICR.** Magnetic ink character recognition.

**module.** In a bar code, the basic element of width. Actual bar code elements can be a module width or a multiple of a module width. The multiple need not be an integer. The smallest element is also called "unit" or "x dimension" in some bar code specifications.

**Multibyte Character Set (MBCS).** A computer representation of a character set in which each character is represented by one or more bytes. A multibyte character set allows many thousands of characters to be represented in a single character set. Common multibyte character sets include PC-mixed, host-mixed and extended UNIX<sup>®</sup> coded character set.

**Multiple Virtual Storage (MVS).** Multiple Virtual Storage, consisting of MVS/System Product Version 1 and the MVS/370 Data Facility Product operating on a System/370<sup>™</sup> processor.

**MVS.** Multiple Virtual Storage.

## N

**no operation (NOP).** A construct whose execution causes a product to do nothing other than to proceed to the next instruction to be processed.

**NOP.** No operation.

**normal duplex printing.** Printing on both sides of the paper so that the sheets can be bound on the long edge of the paper. Contrast with *simplex printing*. See also *tumble duplex printing*.

**numeric data.** (1) Data represented by numerals. (I) (A) (2) Data in the form of numerals and special characters. For example, a date represented as 91/01/01. (3) See also *character data*.



## O

**object.** A collection of structured fields. The first structured field provides a begin-object function and the last structured field provides an end-object function. The object can contain one or more other structured fields whose content consists of one or more data elements of a particular data type. An object can be assigned a name, which can be used to reference the object. Examples of objects are text, font, graphics, and image objects.

**object area.** A rectangular area on a logical page into which a data object is mapped. Examples are graphics block and image block.

**object data.** A collection of related data elements that have been bundled together. Examples of data elements are graphic characters, image data elements, and drawing orders.

**offset stacking.** A function that lets the printed output pages be offset for easy separation of print jobs.

**OGL/370.** Overlay Generation Language/370.

**option.** (1) A specification in a statement that might be used to influence the execution of the statement. (2) A choice offered from a list of possibilities.

**order.** (1) In IOCA, an image data parameter construct. (2) In GOCA, a graphics construct that defines part of a picture or its visual attributes. See also *drawing order*.

**orientation.** The number of degrees an object is rotated relative to a reference; for example, the orientation of an overlay relative to the page origin. Usually applies to blocks of information. Character rotation applies to individual characters. See also *text orientation*.

**origin.** A pel position from which the placement and orientation of text, images, and page segments are specified. For example, pages, overlays, and page segments have origins.

**overlay.** A collection of predefined, constant data such as lines, shading, text, boxes, or logos, that is electronically composed and stored as an AFP resource file that can be merged with variable data on a page while printing or viewing.

**Overlay Generation Language/370 (OGL/370).** An IBM licensed program you can use to design objects for electronic overlays, such as lines, boxes, shadings, and irregular shapes, to create graphics.

## P

**page.** Part of an AFP document bracketed by a pair of Begin Page and End Page structured fields.

**page definition.** A resource used by PSF that defines the rules of transforming line data into composed pages and text controls.

**page format.** Synonym for *data map*.

**page group.** Logical boundaries within documents. For example, a page group can be one customer statement in a file made up of many customer statements.

**page origin.** The point on the logical page that represents  $X_p=0$ ,  $Y_p=0$  in the  $X_p$ ,  $Y_p$  coordinate system. The page origin is relative to the top left of the form. Contrast with *media origin*.

**page overlay.** An electronic overlay that can be invoked for printing and positioned at any point on the page by an Include Page Overlay structured field in the print data. See also *medium overlay* and *overlay*.

**page printer.** A device that prints one page as a unit. (I) (A) Contrast with *line printer*.

**Page Printer Formatting Aid/370 (PPFA/370).** An IBM licensed program that you can use to create and store form definitions and page definitions.

**page segment.** A resource that can contain text and images and can be included on any addressable point on a page or electronic overlay. A page segment assumes the environment of an object in which it is included.

**parameter.** A variable that is given a constant value for a specified application and that can denote the application.

**partial page.** A page that does not contain all the intended data. Partial pages can be printed after an error is sensed.

**pattern.** A symbol used repeatedly to fill an area.

**pel.** Picture element.

**physical page.** Synonymous with *form*, *medium*, and *sheet*. See also *logical page*.

**picture element (pel).** (1) In computer graphics, the smallest element of a display surface that can be independently assigned color and intensity. (T) (2) The addressable unit on a page printer. See also *all-points-addressable* and *raster*.

**pitch.** A unit of width of type, based on the number of characters that can be placed in a linear inch. For example, 10-pitch type has ten characters per inch.

**point.** A unit of about 1/72 inch used in measuring type. Contrast with *pitch*.

**point size.** The height of a font in points.

## Glossary

**portrait.** Pertaining to a display or hardcopy of a page with a greater height than width. Contrast with *landscape*.

**position.** A position in a presentation space or on a presentation surface that can be identified by a coordinate from the coordinate system of the presentation space or presentation surface. See also *picture element*. Synonymous with *addressable position*.

**PPFA/370.** Page Printer Formatting Aid/370.

**ppi.** Pels per inch

**presentation space.** A conceptual address space with a specified coordinate system and a set of addressable positions. The coordinate system and addressable positions can coincide with those of a presentation surface. Examples of presentation spaces are medium, page, and object area.

**presentation text.** (1) Text data and text control information that dictates the format, placement, and appearance of data to be printed. (2) Print data that has been composed into pages. Text formatting programs such as DCF can produce presentation text consisting entirely of structured fields. (3) Synonymous with *composed text*.

**print file.** A file that is created for the purpose of printing data.

**print job.** The data that the user submits to PSF to be printed. A print job can request the printing of multiple data sets.

**print quality.** (1) The measure of printed output against existing standards and in comparison with jobs printed previously. (2) The ability of some page printers to print data at more than one level of print quality, such as “draft” and “near-letter” quality.

**Print Services Facility (PSF).** An IBM licensed program that manages and controls the input data stream and output data stream required by supported AFP printers. PSF combines print data with other resources and printing controls to produce AFP output.

**PSF.** Print Services Facility.

## R

**raster.** Computer graphics in which a display image is composed of an array of pels arranged in rows and columns.

**relative positioning.** Establishing a position within a coordinate system as an offset from the current position. Contrast with *absolute positioning*.

**resolution.** In computer graphics, a measure of the sharpness of an image, expressed as the number of

lines and columns on the display screen or the number of pels per unit of linear measure.

**resource.** An object that contains printing instructions and sometimes data that can be used in a print job. It is a special kind of AFP object that can be stored in a library and can be shared by different printer files and different pages within a printer file. Coded fonts, code pages, font character sets, form definitions, overlays, page definitions, and page segments are AFP resources.

**rotation.** The alignment of a character with respect to its character baseline, measured in degrees in a clockwise direction. Examples are 0°, 90°, 180°, and 270°. 0° character rotation exists when a character is in its customary alignment with the baseline. Synonymous with *character rotation*.

**row.** The horizontal element in a table; it is divided into vertical elements called columns and can be divided horizontally into elements called subrows.

**rule.** A solid or patterned line of any weight, extending horizontally or vertically across a column, row, or page.

## S

**scale.** To enlarge or reduce all or part of a display image.

**scaling.** Making a picture, or part of it, smaller or larger. Scaling is done by multiplying the coordinate values of the picture by a constant amount.

**segment.** A grouping of graphic drawing orders that can appear in a picture.

**semantics.** The part of a construct's description that describes the function of the construct.

**shading.** A darkened area on the displayed page. Usually used to highlight an area containing text. In AFP documents, image data is used to produce shading.

**sheet.** A physical entity on which information is printed. An example of a sheet is one piece of paper. See also *physical sheet*, *medium*, and *form*.

**simplex printing.** Printing on only one side of the paper. Contrast with *duplex printing*.

**single-byte font.** A font having one byte per code point.

**skip.** (1) To ignore one or more instructions in a sequence of instructions. (A) (2) A move of the current print position to another location.

**state.** The portion of the document in which AFP Toolbox is operating, for example, document state or page state.

**structured field.** A self-identifying string of bytes and its data or parameters.

**subset.** Within a hierarchy structure, a portion of architecture represented by a particular level.

**symbol.** (1) A visual representation of something by reason of relationship, association, or convention. (2) In GOCA, the sub-picture referenced as a character definition within a font character set and used as a character, marker, or fill pattern. A bitmap can also be referenced as a symbol for use as a fill pattern.

**syntax.** The part of a construct's description that describes the structure of the construct.

## T

**tag.** A type of structured field used for indexing in an AFP document. Tags associate an index attribute - value pair with a specific page or group of pages in a document.

**text.** A graphic representation of information. Text can consist of alphanumeric characters and symbols arranged in paragraphs, tables, columns, and other shapes.

**text control.** Structured-field data that controls the format, placement, and appearance of text.

**text orientation.** A description of the appearance of text as a combination of inline direction and baseline direction. See also *inline direction* and *baseline direction*.

**trace.** A record of the execution of a computer program. It exhibits the sequences in which the instructions were executed. (A)

**transaction.** In CICS, one of more application programs that can be used by a display station operator. A given transaction can be used concurrently from one or more display stations.

**tumble duplex printing.** Duplex printing for sheets that are to be bound on the short edge of the paper regardless of whether the printing is portrait or landscape. Contrast with *normal duplex printing*.

**TYPE.** A table heading for architecture syntax. The entries under this heading indicate the types of data present in a construct. The data type will be one of the following: BITS, CHAR, CODE, SBIN, UBIN.

**type font.** A collection of characters sharing the same type family, typeface, and size.

**typographic fonts.** Fonts in which the distance between characters varies. The distance from one character to another is adjusted to improve the visual flow of text by eliminating excess space.

## V

**value.** A quantity assigned to a constant, a variable, a parameter, or a symbol in a command.

**variable space.** A method used to assign a character increment dimension of varying size to space characters. The space characters are used to distribute white space within a text line. The white space is distributed by expanding or contracting the dimension of the variable space character's increment, dependent upon the amount of white space to be distributed. See also *variable space character*.

**variable space character.** The code point assigned by the data-stream for which the character increment will be varied according to the semantics and pragmatics of the variable space function. This code point is not presented, but its character increment parameter is used to provide spacing.

**virtual machine (VM).** A functional equivalent of either a System/370 computing system or a System/370-Extended Architecture computing system. Each virtual machine is controlled by an operating system. VM controls concurrent execution of multiple virtual machines on a single system.

**VM.** Virtual machine.

**Virtual Storage Extended (VSE).** An operating system that is an extension of Disk Operating system/Virtual Storage. A VSE system consists of licensed VSE/Advanced Functions support and any programs required to meet the data processing needs of the user. VSE and the hardware it controls form a complete computing system.

**VSE.** Virtual Storage Extended.

## W

**white space.** The portion of a line that is not occupied by characters when the characters of all words that can be placed on a line and spaces between those words are assembled or formatted on a line. When a line is justified, the white space is distributed between the words, characters, or both on the line in some specified manner.

**window.** A predefined part of a graphics presentation space.

## X

**X-axis.** In printing, an axis perpendicular to the direction in which the paper moves through the printer. See also *Y-axis*.

## Glossary

**X-extent.** The width a page or overlay along the X-axis or the size of a page or overlay in the X-direction (horizontal).

**Xm, Ym coordinate system.** The media coordinate system.

**Xp, Yp coordinate system.** The logical page coordinate system.

## Y

**Y-axis.** In printing, an axis parallel with the direction in which the paper moves through the printer. See also *X-axis*.

**Y-extent.** A measurement along the Y-axis.

---

## Bibliography

This bibliography lists the titles of publications containing additional information about Printer Services Facility (PSF), Advanced Function Presentation (FOP), the MVS and other operating systems and related products.

The titles and order numbers may change from time to time. To verify the current title or order number, consult your IBM marketing representative.

You can obtain many of the publications listed in this bibliography from the Printing Systems Digital Library:

<http://www.ibm.com/printers/r5psc.nsf/web/manuals>

---

### Advanced Function Presentation (AFP)

The following publications contain information about IBM's Advanced Function Presentation (AFP) concepts and procedures, and the AFP architecture:

Publication	Order Number
<i>Advanced Function Presentation: Printing Application Development Guide</i>	GC66-0255
<i>Advanced Function Presentation: Application Programming Interface Programming Guide and Reference</i>	S544-3872
<i>Advanced Function Presentation: Printer Information</i>	G544-3290
<i>Advanced Function Presentation: Printer Summary</i>	G544-3135
<i>Advanced Function Presentation: Programming Guide and Line Data Reference</i>	S544-3884
<i>Advanced Function Printing: Host Font Data Stream Reference</i>	S544-3289
<i>AFP Toolbox for Multiple Operating Systems User's Guide</i>	G544-5292
<i>AFP Workbench for Windows: Using the Viewer Application</i>	G544-3813
<i>Guide to Advanced Function Presentation</i>	G544-3876
<i>IBM Page Printer Formatting Aid: User's Guide</i>	S544-5284
<i>IBM Overlay Generation Language/370: User's Guide</i>	S544-3702
<i>Mixed Object Document Content Architecture Reference</i>	SC31-6802
<i>Printing and Publishing Cluster Collection CD-ROM</i>	SK2T-2921
<i>Font Object Content Architecture Reference</i>	S544-3285
<i>Graphic Object Content Architecture Reference</i>	SC31-6804
<i>Image Object Content Architecture Reference</i>	SC31-6805
<i>Intelligent Printer Data Stream Reference</i>	S544-3417
<i>Presentation Text Object Content Architecture Reference</i>	SC31-6803

---

### BCOCA

The following publication contains information about bar code concepts related to the PPFA BARCODE subcommand:

Publication	Order Number
<i>Data Stream and Object Architectures: Bar Code Object Content Architecture Reference</i>	S544-3766

## Bibliography

---

### OS/400 and iSeries

The following publications contain information about OS/400 that uses the form definition and page definitions created by PPFA, and Infoprint Server for iSeries information:

Publication	Order Number
<i>OS/400 Guide to AFP and PSF</i>	S544-5319
<i>OS/400 Printer Device Programming</i>	SC41-3713
<i>OS/400 Data Description Specifications</i>	SC41-962
<i>OS/400 Command Language Reference</i>	SC41-3722
<i>IBM OS/400 printing IV</i>	GG24-4389
<i>OS/400 Advanced Utility User's Guide</i>	S544-5351
<i>IBM Advanced Function Presentation: Toolbox for OS/400 User's Guide</i>	S544-5368
<i>Infoprint Server for iSeries: User's Guide</i>	G544-5775
<i>Infoprint Server for iSeries: Introduction and Planning Guide</i>	G544-5774
<i>Infoprint Designer for iSeries: Getting Started</i>	G544-5773

---

### MVS

The following publications contain information about the general concepts and procedures for the MVS environment:

Publication	Order Number
<i>Print Services Facility/MVS: Application Programming Guide</i>	S544-3673
<i>Print Services Facility/MVS: System Programming Guide</i>	S544-3672

---

### Infoprint Server for OS/390

The following publications contain information about the infoprint server for OS/390:

Publication	Order Number
<i>OS/390 Infoprint Server Customization</i>	G544-5694
<i>OS/390 Infoprint Server Introduction</i>	G544-5696
<i>OS/390 Infoprint Server Messages and Diagnosis</i>	G544-5690
<i>OS/390 Infoprint Server Migration</i>	G544-5697
<i>OS/390 Infoprint Server Operation and Administration</i>	S544-5693
<i>OS/390 Infoprint Server User's Guide</i>	S544-5692

---

### Print Services Facility (PSF) for OS/390

The following publications contain information about the print services facility (PSF) for OS/390:

Publication	Order Number
<i>IBM IP Printway Guide</i>	S544-5379
<i>IBM NetSpool Guide</i>	G544-5301
<i>PSF for AIX: Upload Configuration Guide for SNA</i>	S544-5422



<i>PSF for AIX: Upload Configuration Guide for TCP/IP</i>	S544–5423
<i>PSF for OS/390 Collection Kit CD-ROM</i>	SK2T-9267
<i>PSF for OS/390: Customization</i>	S544–5622
<i>PSF for OS/390: Diagnosis</i>	G544–5623
<i>PSF for OS/390: Download for OS/390</i>	S544–5624
<i>PSF for OS/390: Introduction</i>	S544–5625
<i>PSF for OS/390: Licensed Program Specifications</i>	G544–5626
<i>PSF for OS/390: Messages and Codes</i>	G544–5627
<i>PSF for OS/390: User's Guide</i>	S544–5630
<i>AFP Conversion and Indexing Facility: User's Guide</i>	S544–5285
<i>PSF: Security Guide</i>	S544–3291
<i>Program Directory for Print Services Facility for OS/390</i>	None
<i>Program Directory for Download for OS/390</i>	None
<i>Program Directory for IP PrintWay Feature of PSF for OS/390</i>	None
<i>Program Directory for IP NetSpool Feature of PSF for OS/390</i>	None

---

## Fonts

The following publications contain information about the fonts used by IBM:

<b>Publication</b>	<b>Order Number</b>
<i>ABOUT TYPE: IBM's Technical Reference for 240-pel Digitized Type</i>	S544–3516
<i>ABOUT TYPE: IBM's Technical Reference for Core Interchange Digitized Type</i>	G544–3708
<i>IBM AFP Fonts: Font Samples</i>	S544–3792
<i>IBM AFP Fonts: Font Summary</i>	G544–3810
<i>IBM AFP Fonts: Font Summary for AFP Font Collection</i>	S544–5633
<i>IBM AFP Fonts: IBM's Typographic Primer for Digitized Type</i>	G544–3183
<i>IBM AFP Fonts: Introduction to Typography</i>	G544–3122
<i>IBM AFP Fonts: Technical Reference for Code Pages</i>	S544–3802
<i>IBM AFP Fonts: Technical Reference for IBM Chinese, Japanese, and Korean Fonts</i>	S544–5330
<i>IBM AFP Fonts: Technical Reference for IBM Expanded Core Fonts</i>	S544–5625
<i>IBM AFP Fonts: Type Transformer User's Guide</i>	G544–3796

---

## Text Processing

The following publications contain information about text processing:

<b>Publication</b>	<b>Order Number</b>
<i>DCF/DLF General Information</i>	GH20–9158
<i>Document Composition Facility: Bar Code User's Guide</i>	S544–3115
<i>Document Composition Facility: SCRIPT/VS Text Programmer's Guide</i>	SH35–0069
<i>Publishing Systems BookMaster General Information</i>	GC34–5006
<i>Publishing Systems BookMaster User's Guide</i>	SC34–5009

## Bibliography

<i>Using DisplayWrite/370</i>	SH12–5172
<i>IBM AFP Fonts: Technical Reference for Code Pages</i>	S544–3802
<i>IBM AFP Fonts: Technical Reference for IBM Chinese, Japanese, and Korean Fonts</i>	S544–5330
<i>IBM AFP Fonts: Technical Reference for IBM Expanded Core Fonts</i>	S544–5625
<i>IBM AFP Fonts: Type Transformer User's Guide</i>	G544–3796

---

## Infoprint Manager

The following publications contain information about IBM's Infoprint Manager:

Publication	Order Number
<i>IBM Infoprint Manager: Reference</i>	S544–5475
<i>IBM Infoprint Manager for AIX: Administrator's Guide</i>	S544–5595
<i>IBM Infoprint Manager for AIX: User's and Operator's Guide</i>	S544–5596
<i>IBM Infoprint Manager for Windows NT and Windows 2000: Planning Guide</i>	G544–5716
<i>IBM Infoprint Manager for Windows NT and Windows 2000: Getting Started</i>	G544–5717
<i>IBM Infoprint Manager for Windows NT and Windows 2000: Configuration Guide</i>	Web-Based
<i>IBM Infoprint Manager for Windows NT and Windows 2000: PSF Direct Network Configuration Guide</i>	Web-Based

---

## Printers

The following publications contain information about IBM's printers:

Publication	Order Number
<i>Reference Manual for the IBM 3800 Printing Subsystem Models 3 and 6</i>	GA32–0050
<i>IBM PagePrinter 3812 Introduction and Planning Guide</i>	G544–3265
<i>IBM 3816 Page Printer Operating Instructions</i>	GA34–2075
<i>IBM 3825 Page Printer Product Description</i>	G544–3482
<i>IBM 3827 Page Printer Product Description</i>	G544–3194
<i>IBM 3828 Advanced Function MICR Printer Product Description</i>	G544–3361
<i>IBM 3900 Advanced Function Printer Product Description</i>	GA32–0135
<i>IBM 3912 and 3916 Page Printer Getting Started</i>	S544–3898
<i>IBM LaserPrinter 4028 Introduction and Planning Guide</i>	S544–4258
<i>User's Guide for the IBM LaserPrinter 4029 Series</i>	ZA40–0542
<i>IBM 4224 Printer Models 1xx and 2xx Product and Programming Description Manual</i>	GC31–2551
<i>IBM 4230 Printer Product and Programming Description Models 102 and 202</i>	GC40–1701
<i>IBM 4234 Printer Models 007, 008, 011 and 012 Product and Programming Description</i>	GC31–3879
<i>InfoPrint 60, 3130, 3160, and 3935 Advanced Function Printer Attachment Configuration Handbook</i>	S544–3977
<i>InfoPrint 60 Finisher Application Design Guide</i>	S544–5643
<i>InfoPrint 62 Introduction and Planning Guide</i>	G544–5384
<i>InfoPrint 3000 Introduction and Planning Guide</i>	G544–5563
<i>InfoPrint 4000 and 3900 Advanced Function Printers Introduction and Planning Guide</i>	G544–5427



<i>InfoPrint Color 100 Introduction and Planning Guide</i>	G544–5612
<i>InfoPrint Hi-Lite Color Introduction and Planning Guide</i>	G544–5420
<i>IOCP and ESCON Reference</i>	GC38–0401

---

## TCP/IP

The following publications contain information about TCP/IP:

Publication	Order Number
<i>Internetworking with TCP/IP, Principles, Protocols and Architecture</i>	SC31–6144
<i>TCP/IP Tutorial and Technical Reference</i>	GG24–3376
<i>InfoPrint Hi-Lite Color Introduction and Planning Guide</i>	G544–5420

---

## TCP/IP for MVS

The following publications contain information about TCP/IP for MVS:

Publication	Order Number
<i>TCP/IP for MVS: Customization and Administration Guide</i>	SC31–7134
<i>TCP/IP for MVS: Application Programming Interface Reference</i>	SC31–7187
<i>TCP/IP for MVS: Programmer's Reference</i>	SC31–7135

---

## VTAM and NCP

The following publications contain information about VTAM and NCP:

Publication	Order Number
<i>ACF/INCP/SSP Version 3 Resource Reference</i>	SC30–3254
<i>Advanced Communications Function for VTAM, Version 3, Programming</i>	SC23–0115
<i>Advanced Communications Function for VTAM, Version 2, Programming</i>	SC27–0611
<i>Advanced Communications Function for VTAM, Version 3, Customization</i>	SC23–0112
<i>NCP Resource Definition Guide</i>	SC30–3449
<i>NCP: Resource Definition Reference</i>	SC30–3254
<i>NCP: System Support</i>	SC30–3447
<i>Network Program Products: Bibliography and Master Index</i>	GC30–3353
<i>Network Program Products: General Information</i>	GC23–0108
<i>VTAM Resource Definition Reference</i>	SC31–6552
<i>VTAM Version 3 Diagnosis Reference</i>	LY30–5582
<i>VTAM Version 3 Messages and Codes</i>	SC31–6433
<i>VTAM Version 3 Operation</i>	SC23–0113
<i>VTAM Version 4 Diagnosis Guide</i>	LX75–0204
<i>VTAM Version 4 Diagnosis Quick Reference</i>	LX75–0203
<i>VTAM Version 4 Messages and Codes</i>	SC31–6418

## Bibliography

---

### System Network Architecture (SNA)

The following publications contain information about System Network Architecture:

Publication	Order Number
<i>SNA Customization</i>	LY43-0110
<i>SNA Formats (SNA Reference Summary)</i>	GA27-3136
<i>SNA Resource Definition Reference</i>	SC31-8565
<i>SNA Technical Overview</i>	GC30-3073
<i>SNA transaction Programmer's Reference Manual for LU Type 6.2</i>	GC30-3084
<i>Systems Network Architecture Formats</i>	GA27-3136

Several other publications may help you understand the licensed programs used with the AFP Toolbox. Publications can be ordered from IBM by the order number.

---

### IBM Architecture Publications

Publication	Order Number
<i>MO:DCA Reference</i>	SC31-6802
<i>Image Object Content Architecture Reference</i>	SC31-6805
<i>Graphics Object Content Architecture Reference</i>	SC31-6804
<i>Presentation Text Object Content Architecture Reference</i>	SC31-6803
<i>Bar Code Object Content Architecture Reference</i>	S544-3766

---

### IBM Advanced Function Presentation Publications

Publication	Order Number
<i>IBM OnDemand for AIX: Introduction and Planning Guide</i>	G544-5281

---

# Index

## A

- AddBarCode, BCOCA Class 344
- AddDbText, PText Class 328
- adding data to bar codes 62
- AddPage
  - AFPDclass Class 315
- AddText, PText Class 327
- advance line 183
- AESstring (for C++) 310
- AFP
  - data and resource objects 9
  - hierarchy 4
  - structured fields 6
- AFP Font Collection
  - AIX 390
  - AS/400 397
  - MVS environment 375
  - OS/400 environment 395
- AFP Toolbox calls 53, 54
  - page-level 55
  - sequence 53
- AFP Toolbox Table Samples 405
- AFP Workbench for OS/2 and Windows
  - viewing online documentation (AIX) 389
- AFPDMP (AIX) 390
- AFPDclass Class 314
  - AddPage 315
  - BeginGroup 315
  - Comment 316
  - EndGroup 316
  - InvokeMediumMap 316
  - Link 316
  - Mnemonic 316
  - PreloadObject 316
  - PreloadOverlay 317
  - SetDocumentCodepage 314
  - SetInputCodepage 314
  - SetInputDbCodeset 315
  - Tag 317
  - TimeStamp 317
- AFPPage Class 319
  - Comment 320
  - CopyPage 320
  - IncludeBarCode 320
  - IncludeBarCodeInline 320
  - IncludeGraphic 321
  - IncludeGraphicInline 321
  - IncludeImage 321
  - IncludeImageInline 321
  - IncludeObject 321
  - IncludeOverlay 323
  - IncludePageSegment 323
  - IncludeText 324
  - KeepObject 324
  - Link 324
  - Mnemonic 324
  - PageName 324

- AFPPage Class (*continued*)
  - PageSize 324
  - RotateOverlay 325
  - SetObjectColorProfile 325
  - ShadeBlock 325
  - Tag 325
- AIX environment
  - AFP Font Collection 390
  - compiling sample programs 389
  - displaying list of fonts 390
  - displaying structured field contents 390
  - FontMeister directory 389
  - running sample programs 389
  - tools 390
    - AFPDMP 390
    - FLIPLIST 390
  - troubleshooting 393
  - viewing online documentation 389
- archiving documents
  - Begin Group 84
  - description 12
  - End Group 139
  - Put Tag 201
- arcs
  - GOCA Begin Area 267
  - GOCA Partial Circle 282
  - GOCA Partial Ellipse 284
- areas
  - BeginArea (C++) 347
  - EndArea (C++) 349
- AS/400 environment
  - AFP Font Collection 397
  - compilers 395
  - prerequisites 395
  - system requirements 395
- attributes
  - Define Double-Byte Font By Attribute procedure call 114
  - Define Font By Attribute procedure call 120
  - Define Font By Attributes With Scaling procedure call 123
  - indexing 201

## B

- Bar Code Data procedure call 62
- bar codes
  - AddBarCode (C++) 344
  - Bar Code Data 62
  - Begin Bar Code 66
  - C++ 342
  - Data Matrix Bar Code Data 111
  - description 33
  - End Bar Code 134
  - Export (C++) 345
  - IncludeBarCode (C++) 320
  - IncludeBarCodeInline (C++) 320
  - MaxiCode Bar Code Data 175

- bar codes *(continued)*
  - PDF417 Bar Code Data 186
  - working with 33
- BCOCA Class 342
  - AddBarCode 344
  - CopyOca 345
  - Export 345
  - Font 345
  - Mnemonic 345
- Begin Bar Code procedure call 66
- Begin Box procedure call 73
- Begin Color procedure call 75
- Begin Document procedure call 78
- Begin Field procedure call 82
- Begin Graphic Object 263
- Begin Group procedure call 84
- Begin Image procedure call 86
- Begin Page procedure call 91
- Begin Paragraph procedure call 93
- Begin Row procedure call 96
- Begin Rule procedure call 97
- Begin Session procedure call 100
- Begin Shade procedure call 101
- Begin Table procedure call 103
- Begin Underscore procedure call 106, 149
- BeginArea, GOCA Class 347
- BeginGroup
  - AFPDoc Class 315
- BeginImage, IOCA Class 339
- beginning a document 15, 18
- beginning a page 21
- beginning a session 16
- Bibliography Section 485
- BOOLEAN 60
- boxes
  - Begin Box 73
  - DrawBox (C++) 329, 348
  - drawing 24
  - End Box 135
  - GOCA Begin Area 267
  - GOCA Draw Box 272
  - GOCA Set Line Parameters 294
  - GOCA Set Pattern Symbol 298
  - Put Box 192
- buffers
  - Get Buffer 150
- building an AFP document (C++) 309

## C

- C Language Reference 53
- C library 6
  - FontMeister 6
- C sample notes 15
- C++ language programs
  - AEStrng 310
  - building an AFP document 309
  - including header files
    - AIX system 311
    - MVS system 311
- C++ Object Library 5

- C++ Object Library *(continued)*
  - C++ language programs 309
- C++ objects 261, 309
- C++ sample notes 15
- CBLGetBuffer Program
  - example 382
- characters
  - angles and directions 288
  - GOCA Set Process Color 301
  - SetCharAngle (C++) 350
  - SetCharCell (C++) 350
  - SetCharDir (C++) 350
  - SetCharPrec (C++) 350
- CharString, GOCA Class 348
- CICS not supported
  - MVS environment 375
- circles
  - FullArc (C++) 349
  - GOCA Circle 271
  - GOCA Partial Circle 282
  - GOCA Set Line Parameters 294
  - GOCA Set Pattern Symbol 298
  - SetArcParms (C++) 350
- classes (C++)
  - AFPDoc 314
  - AFPPage 319
  - BCOCA 342
  - Document 312
  - GOCA 346
  - IOCA 337
  - OCA 333
  - Page 318
  - PText 326
- COBOL Language Reference 53
- COBOL library 6
- COBOL sample notes 15
- code pages
  - description 44
  - Set Data Stream Codepage 217
  - Set Input Codepage 225
  - Set Text Codepage 236
  - working with 44
- codesets
  - Set Input Double-Byte Codeset 228
  - setting for double-byte text 238
- color
  - Begin Color 75
  - Color (C++) 328
  - End Color 136
  - GOCA Set Extended Color 292
  - GOCA Set Process Color 301
  - Put Color 194
  - querying 209
  - Set Color 215
  - Set Extended Color 219
  - SetExtColor (C++) 351
  - SetProcessColor (C++) 352
- color profiles
  - Set Object Color Profile 232
  - SetObjectColorProfile (C++) 325
- Color, PText Class 328

- Comment
  - AFPDClass Class 316
  - AFPPage Class 320
- comments
  - Comment (C++) 316
  - Output Comment 184
- compilers
  - AS/400 environment 395
  - MVS environment 375
- compiling sample programs
  - AIX 389
  - MVS 376
  - OS/400 395
- coordinate systems
  - logical page 7
  - medium 7
  - physical page 7
- copy books, GOCA 262
- CopyOca, BCOCA Class 345
- CopyOca, IOCA Class 339
- CopyOCA, OCA Class 334
- CopyPage
  - AFPPage Class 320
  - Page Class 318
- Create Link procedure call 108
- creating
  - pages 20
  - paragraphs 26
  - strings 22

## D

- data 9
- Data Matrix Bar Code Data procedure call 111
- data objects created by Toolbox 10
- data types
  - BOOLEAN 60
  - FontID 60
  - HANDLE 60
- Define Double-Byte Font By Attribute
  - C++ program call 356
- Define Double-Byte Font By Attribute procedure call 114
- Define Field procedure call 117
- Define Font By Attribute procedure call 120
- Define Font By Attribute With Scaling C++ program call 360
- Define Font By Attributes C++ program call 358
- Define Font By Attributes With Scaling procedure call 123
- Define Font By Name C++ program call 362
- Define Font By Name procedure call 126
- Define Row procedure call 128
- defining fonts 17
- Delete Font procedure call 131
- Delete Page procedure call 132
- determining page breaks 28
- displaying a list of fonts
  - AIX 390
- displaying structured field contents
  - AIX 390

- DocName
  - Document Class 312
- Document Class 312
  - DocName 312
  - Mnemonic 312
  - NumPages 313
  - OutputThreshold 313
- document-level calls 54
- documents
  - AFPDClass Class 314
  - Begin Document 78
  - beginning 18
  - C++ 312
  - End Document 137
  - ending 31
  - indexing 19
- double-byte codesets
  - Set Input Double-Byte Codeset 228
- double-byte fonts
  - Define Double-Byte Font By Attribute 114
  - Define Double-Byte Font By Attribute (C++) 356
- double-byte strings
  - Measure Double-Byte String 178
  - Measure Double-Byte String (C++) 363
  - SetInputDbCodeset (C++) 315
  - Translate And Measure Double-Byte String 244
  - Translate And Measure Double-Byte String (C++) 368
  - Write Double-Byte String 253
- double-byte text
  - AddDbText (C++) 328
  - Put Double-Byte Text 196
- DrawBox, GOCA Class 348
- DrawBox, PText Class 329
- DrawFillet, GOCA Class 348
- drawing
  - boxes 24, 73, 192
  - DrawBox (C++) 329
  - DrawBox(C++) 348
  - DrawFillet (C++) 348
  - DrawMarkers (C++) 349
  - DrawRule (C++) 329
  - DrawRules (C++) 349
  - FullArc (C++) 349
  - GOCA Draw Objects 274
  - GOCA No Operation 281
  - GOCA Set Line Parameters 294
  - RelativeRules (C++) 349
  - rules 24
  - SetArcParms (C++) 350
- DrawMarkers, GOCA Class 349
- DrawRule, PText Class 329
- DrawRules, GOCA Class 349

## E

- ellipses
  - FullArc (C++) 349
  - GOCA Ellipse 277
  - GOCA Partial Ellipse 284
  - GOCA Set Line Parameters 294

- ellipses (*continued*)
  - GOCA Set Pattern Symbol 298
  - illustration 277
  - SetArcParms (C++) 350
- End Bar Code procedure call 134
- End Box procedure call 135
- End Color procedure call 136
- End Document procedure call 137
- End Field procedure call 138
- End Graphic Object 266
- End Group procedure call 139
- End Image procedure call 140
- End Page procedure call 141
- End Paragraph procedure call 142
- End Row procedure call 143
- End Rule procedure call 145
- End Session procedure call 146
- End Shade procedure call 147
- End Table procedure call 148
- EndArea, GOCA Class 349
- EndGroup
  - AFPDclass Class 316
- EndImage, IOCA Class 339
- ending a document 31
- ending a page 29
- ending a session 32
- environment variables
  - OS/390 Unix System Services 385
- example document
  - elements of 13
  - getting started 15
- example programs
  - C (MVS) 378
  - C (OS/400) 396
  - C++ (MVS) 380
  - CBLGetBuffer program (MVS) 382
  - COBOL (MVS) 376
  - COBOL (OS/400) 396
  - RPG (OS/400) 397
- examples
  - beginning a document 18
  - beginning a page 21
  - creating a paragraph 26
  - creating pages 20
  - defining fonts 17
  - determining page breaks 28
  - drawing a box 24
  - drawing a rule 24
  - ending a document 31
  - ending a page 29
  - ending a session 32
  - including a resource 25
  - indexing 19
  - placing text 22
  - putting data on the page 22
  - working with bar codes 33
  - working with tables 36
- Export
  - BCOCA Class 345
- Export, IOCA Class 339

- extended color
  - Set Extended Color 219
  - SetExtColor (C++) 351

## F

- fields
  - Begin Field 82
  - Define Field 117
  - End Field 138
- fill patterns 298
  - GOCA Set Pattern Symbol 298
- fillets
  - described 275
  - DrawFillet (C++) 348
  - GOCA Begin Area 267
  - GOCA Draw Objects 274
- FLIPLIST (AIX) 390
- Font, BCOCA Class 345
- FontID 60
- FontMeister
  - C library 6
  - directory (AIX) 389
  - DLL location (AIX) 389
- fonts 10
  - Define Double-Byte Font By Attribute 114
  - Define Font By Attribute 120
  - Define Font By Attribute (C++) 358
  - Define Font By Attributes With Scaling 123
  - Define Font By Name 126
  - Define Font By Name (C++) 362
  - defining 17
  - Delete Font 131
  - efine Font By Attribute With Scaling (C++) 360
  - Font (C++) 345
  - GOCA Set Font 290
  - querying 209
  - return codes 259
  - Set Font 221
  - Set Font Type 223
  - Set Font Type (C++) 367
  - SetFont (C++) 330
  - updating your font index (AIX) 390, 397
  - using 46
- form definitions
  - description 11
- format of function call descriptions 60
- FullArc, GOCA Class 349

## G

- Get Buffer procedure call 150
- getting started 15
  - sample document 15
- GOCA Begin Area 267
- GOCA Character String 269
- GOCA Circle 271
- GOCA Class 346
  - BeginArea 347
  - CharString 348
  - DrawBox 348

- GOCA Class *(continued)*
  - DrawFillet 348
  - DrawMarkers 349
  - DrawRules 349
  - EndArea 349
  - FullArc 349
  - NoOP 349
  - QueryGPSPos 349
  - RelativeRules 349
  - SetArcParms 350
  - SetCharAngle 350
  - SetCharCell 350
  - SetCharDir 350
  - SetCharPrec 350
  - SetCharSet 351
  - SetExtColor 351
  - SetGPSPos 351
  - SetLineType 351
  - SetLineWidth 351
  - SetMarkerPrec 351
  - SetMarkerSym 351
  - SetPatternSym 352
  - SetProcessColor 352
- GOCA Draw Box 272
- GOCA Draw Objects 274
- GOCA Ellipse 277
- GOCA End Area 280
- GOCA functions 261
  - C and COBOL function descriptions 261
  - C++ objects 261
  - header files and copy books 262
- GOCA No Operation 281
- GOCA Partial Circle 282
- GOCA Partial Ellipse 284
- GOCA presentation space
  - description 347
- GOCA Query GPS Position 286
- GOCA Set Character Parameters 287
- GOCA Set Extended Color 292
- GOCA Set Font 290
- GOCA Set GPS Position 291
- GOCA Set Line Parameters 294
- GOCA Set Marker Parameters 296
- GOCA Set Pattern Symbol 298
- GOCA Set Process Color 301
- GPS
  - query (C++) 349
  - SetGPSPos (C++) 351
- Graphic Inline procedure call 152
- graphic presentation space 264
  - concepts 261
  - GOCA Set GPS Position 291
- graphics
  - Begin Graphic Object 263
  - BeginArea (C++) 347
  - End Graphic Object 266
  - EndArea (C++) 349
  - GOCA Begin Area 267
  - GOCA Draw Objects 274
  - GOCA End Area 280
  - GOCA Partial Circle 282

- graphics *(continued)*
  - GOCA Partial Ellipse 284
  - GOCA Set Pattern Symbol 298
  - Graphic Inline 152
  - IncludeGraphic (C++) 321
  - IncludeGraphicInline (C++) 321
  - NoOP (C++) 349
- group-level tag 12
- grouping 12
- groups
  - Begin Group 84
  - BeginGroup (C++) 315
  - End Group 139
  - EndGroup (C++) 316

## H

- HANDLE 60
- handles 15, 59
- header files
  - including 61
  - including for C++ 311
- header files, GOCA 262
- hierarchy of calls 53
- Hmove, PText Class 329
- HMoveTo, PText Class 329
- Horizontal Move procedure call 154
- Horizontal Move To 23
- Horizontal Move To procedure call 156

## I

- IDESize, IOCA Class 339
- IDEStructure, IOCA Class 339
- Image Data procedure call 158
- Image Inline procedure call 160
- ImageData, IOCA Class 340
- ImageEncoding, IOCA Class 340
- images
  - Begin Image 86
  - BeginImage (C++) 339
  - C++ 337
  - End Image 140
  - EndImage (C++) 339
  - GOCA Set Process Color 301
  - IDESize (C++) 339
  - IDEStructure (C++) 339
  - Image Data 158
  - Image Inline 160
  - ImageData (C++) 340
  - ImageEncoding (C++) 340
  - IncludelImage (C++) 321
  - IncludelImageInline (C++) 321
- Include Object procedure call 162
- Include Overlay procedure call 167
- Include Page Segment procedure call 169
- IncludeBarCode
  - AFPPage Class 320
- IncludeBarCodeInline
  - AFPPage Class 320



- IncludeGraphic
  - AFPPage Class 321
- IncludeGraphicInline
  - AFPPage Class 321
- IncludeImage
  - AFPPage Class 321
- IncludeImageInline
  - AFPPage Class 321
- IncludeObject
  - AFPPage Class 321
- IncludeOverlay
  - AFPPage Class 323
- IncludePageSegment
  - AFPPage Class 323
- IncludeText, AFPPage Class 324
- including
  - C++ header files 311
  - header files 61
- including resources 25
- indexing 19
  - for softcopy 12
  - Put Tag 201
- installing
  - FontMeister directory (AIX) 389
- Invoke Medium Map procedure call 171
- InvokeMediumMap
  - AFPDclass Class 316
- IOCA Class 337
  - BeginImage 339
  - CopyOca 339
  - EndImage 339
  - Export 339
  - IDESize 339
  - IDEStructure 339
  - ImageData 340
  - ImageEncoding 340
  - LookUpTable 340
  - MaxIPDLength 340
  - Mnemonic 340
- IsValid
  - Page Class 318

## K

- Keep Object procedure call 173
- KeepObject
  - AFPPage Class 324

## L

- Language Reference
  - C, COBOL, and RPG 53
- language reference
  - C++ 309
- lines
  - GOCA Begin Area 267
  - GOCA Draw Objects 274
  - GOCA Set Process Color 301
  - Next Line 183
  - SetLineType (C++) 351
  - SetLineWidth (C++) 351

- Link
  - AFPDclass Class 316
- Link, AFPPage Class 324
- linking
  - described 12
- links
  - Create Link 108
  - Link (C++) 316, 324
- logical page 7
- LookUpTable, IOCA Class 340

## M

- markers
  - DrawMarkers (C++) 349
  - GOCA Draw Objects 274
  - GOCA Set Marker Parameters 296
  - GOCA Set Process Color 301
  - SetMarkerPrec (C++) 351
  - SetMarkerSym (C++) 351
- MaxiCode Bar Code Data procedure call 175
- MaxIPDLength, IOCA Class 340
- MaxPTXLength, PText Class 329
- Measure Double-Byte String
  - C++ program call 363
  - procedure call 178
- Measure String
  - C++ program call 365
  - procedure call 180
- media
  - Set Media Size 230
- medium
  - coordinate system 7
- medium maps
  - Invoke Medium Map 171
  - InvokeMediumMap (C++) 316
- migrating from AFP API to AFP Toolbox 401
- Mnemonic
  - AFPDclass Class 316
  - AFPPage Class 324
  - BCOCA Class 345
  - Document Class 312
  - IOCA Class 340
  - OCA Class 334
  - Page Class 318
  - PText Class 330
- MO:DCA document
  - example 3
- MODCA Color 353
- move
  - Hmove (C++) 329
  - HMoveTo (C++) 329
  - Horizontal Move 154
  - Horizontal Move To 156
  - Move (C++) 330
  - MoveTo (C++) 330
  - Vertical Move 249
  - Vertical Move To 251
  - Vmove (C++) 331
  - VMoveTo (C++) 332
- Move, PText Class 330



- MoveTo, PText Class 330
- MVS environment
  - AFP Font Collection 375
  - compilers 375
  - compiling sample programs 376
  - prerequisites 375
  - running sample programs 376
  - system requirements 375
  - troubleshooting 387
  - using fonts with the AFP Toolbox 375

## N

- naming pages 91
- navigating through a soft-copy document 12
- Next Line 23
- Next Line procedure call 183
- NoOP, GOCA Class 349
- NumPages
  - Document Class 313

## O

- Object Library
  - C++ 5
- ObjectAreaContentPosition, OCA Class 334
- ObjectAreaMeasurementUnits, OCA Class 334
- ObjectAreaMixingFlag, OCA Class 334
- ObjectAreaPosition, OCA Class 335
- ObjectAreaSize, OCA Class 335
- ObjectContentMapping, OCA Class 335
- objects 9
  - Include Object 162
  - IncludeObject (C++) 321
  - Keep Object 173
  - KeepObject (C++) 324
  - positioning 335
  - Preload Object 189
  - PreloadObject (C++) 316
  - Set Object Color Profile 232
  - SetObjectColorProfile (C++) 325
- OCA
  - CopyOca (C++) 339
- OCA Class 333
  - CopyOCA 334
  - Mnemonic 334
  - ObjectAreaContentPosition 334
  - ObjectAreaMeasurementUnits 334
  - ObjectAreaMixingFlag 334
  - ObjectAreaPosition 335
  - ObjectAreaSize 335
  - ObjectContentMapping 335
  - OCAType 336
- OCAType, OCA Class 336
- online documentation
  - for AIX
    - viewing 389
- Open Edition system
  - running MVS Toolbox on 385
- orientation
  - illustration 331

- orientation (*continued*)
  - Set Text Orientation 240
  - SetTextOrientation (C++) 330
- OS/390 Unix System Services
  - environment variables 385
- OS/400 environment
  - AFP Font Collection 395
  - compiling sample programs 395
  - running sample programs 395
  - troubleshooting 399
  - using fonts with the AFP Toolbox 395
- Output Comment procedure call 184
- OutputThreshold
  - Document Class 313
- overlays
  - description 11
  - Include Overlay 167
  - IncludeOverlay (C++) 323
  - Preload Overlay 191
  - PreloadOverlay (C++) 317
  - Rotate Overlay 213
  - RotateOverlay (C++) 325
- overview 3

## P

- Page Class 318
  - CopyPage 318
  - IsValid 318
  - Mnemonic 318
- page definitions 11
- page segments 11
  - Include Page Segment 169
  - IncludePageSegment (C++) 323
- page-level calls 55
- page-level tag 12
- PageName, AFPPage Class 324
- pages
  - AddPage (C++) 315
  - Begin Page 91
  - beginning 21
  - C++ 318, 319
  - CopyPage (C++) 318, 320
  - creating 20
  - Delete Page 132
  - End Page 141
  - ending 29
  - indexing 12
  - NumPages (C++) 313
  - PageName (C++) 324
  - PageSize (C++) 324
- PageSize, AFPPage Class 324
- paragraphs 26
  - Begin Paragraph 93
  - creating 26
  - End Paragraph 142
- patterns
  - GOCA Set Process Color 301
  - SetPatternSym (C++) 352
- PDF417 Bar Code Data procedure call 186
- physical page 7

- physical page coordinate system 7
- position
  - GOCA Query GPS Position 286
  - ObjectAreaContentPosition (C++) 334
  - ObjectAreaMeasurementUnits (C++) 334
  - ObjectAreaPosition (C++) 335
  - objects, illustration 335
  - querying 209
  - Set Position 234
  - SetGPSPos (C++) 351
- positioning information, providing 59
- Preload Object procedure call 189
- Preload Overlay procedure call 191
- PreloadObject
  - AFPDClass Class 316
- PreloadOverlay
  - AFPDClass Class 317
- prerequisites
  - AS/400 environment 395
  - MVS environment 375
- printable area
  - setting 230
- printing the output file 47
- providing positioning and size information 59
- PText Class 326
  - AddDbText 328
  - AddText 327
  - Color 328
  - DrawBox 329
  - DrawDrawRuleBox 329
  - Hmove 329
  - HMoveTo 329
  - MaxPTXLength 329
  - Mnemonic 330
  - Move 330
  - MoveTo 330
  - RepeatString 330
  - SetFont 330
  - SetTextOrientation 330
  - Vmove 331
  - VMoveTo 332
- Put Box procedure call 192
- Put Color procedure call 194
- Put Double-Byte Text procedure call 196
- Put Horizontal Rule 24
- Put Horizontal Rule procedure call 198
- Put Shade procedure call 200
- Put Tag procedure call 201
- Put Text in Table Field procedure call 205
- Put Text procedure call 203
- Put Vertical Rule procedure call 207
- putting data on the page 22

## Q

- query
  - DocName (C++) 312
  - GOCA Query GPS Position 286
  - QueryGPSPos (C++) 349
- Query procedure call 209
- QueryGPSPos, GOCA Class 349

## R

- RelativeRules, GOCA Class 349
- Repeat String procedure call 211
- RepeatString, PText Class 330
- resources 9
  - AFP 10
  - generating 9
  - including 25
- return codes
  - description 58
  - font 259
- returning a buffer 150
- Rotate Overlay procedure call 213
- RotateOverlay, AFPPage Class 325
- rotating text 240
- rows
  - Begin Row 96
  - Define Row 128
  - End Row 143
- RPG iInterface 6
- RPG Language Reference 53
- rules
  - Begin Rule 97
  - drawing 24
  - DrawRule (C++) 329
  - DrawRules (C++) 349
  - End Rule 145
  - Put Horizontal Rule 198
  - Put Vertical Rule 207
  - RelativeRules (C++) 349
- running sample programs
  - AIX 389
  - MVS 376
  - OS/400 395

## S

- sample document
  - elements of 13
  - getting started 15
  - putting data on the page 22
- sample programs
  - C (MVS) 378
  - C (OS/400) 396
  - C++ (MVS) 380
  - CBLGetBuffer program (MVS) 382
  - COBOL (MVS) 376
  - COBOL (OS/400) 396
  - compiling (AIX) 389
  - compiling (OS/400) 395
  - RPG (OS/400) 397
  - running (AIX) 389
  - running (OS/400) 395
- session
  - Begin Session 100
  - beginning 16
  - End Session 146
  - ending 32
- session-level calls 54
- Set Color procedure call 215

- Set Data Stream Codepage procedure call 217
- Set Extended Color procedure call 219
- Set Font procedure call 221
- Set Font Type C++ program call 367
- Set Font Type procedure call 223
- Set Input Codepage procedure call 225
- Set Input Double-Byte Codeset procedure call 228
- Set Media Size procedure call 230
- Set Object Color Profile procedure call 232
- Set Position 23
- Set Position procedure call 234
- Set Text Codepage procedure call 236
- Set Text Double-Byte Codeset procedure call 238
- Set Text Orientation procedure call 240
- Set Units procedure call 242
- SetArcParms, GOCA Class 350
- SetCharAngle, GOCA Class 350
- SetCharCell, GOCA Class 350
- SetCharDir, GOCA Class 350
- SetCharPrec, GOCA Class 350
- SetCharSet, GOCA Class 351
- SetDocumentCodepage
  - AFPDClass Class 314
- SetExtColor, GOCA Class 351
- SetFont, PText Class 330
- SetGPSPos, GOCA Class 351
- SetInputCodepage
  - AFPDClass Class 314
- SetInputDbCodeset
  - AFPDClass Class 315
- SetLineType, GOCA Class 351
- SetLineWidth, GOCA Class 351
- SetMarkerPrec, GOCA Class 351
- SetMarkerSym, GOCA Class 351
- SetObjectColorProfile
  - AFPPage Class 325
- SetPatternSym, GOCA Class 352
- SetProcessColor, GOCA Class 352
- SetTextOrientation, PText Class 330
- setting the output position 234
- setting the printable area 230
- shade
  - Begin Shade 101
  - End Shade 147
  - Put Shade 200
  - ShadeBlock (C++) 325
- ShadeBlock, AFPPage Class 325
- size
  - ObjectAreaSize (C++) 335
- size information, providing 59
- starting a document 15
- strings
  - creating 22
  - description 22
  - GOCA Character String 269
  - Measure Double-Byte String 178
  - Measure String 180
  - Measure String (C++) 365
  - Repeat String 211
  - RepeatString (C++) 330
  - SetDocumentCodepage (C++) 314

- strings (*continued*)
  - SetInputCodepage (C++) 314
  - Translate And Measure Double-Byte String 244
  - Translate And Measure String 246
  - Translate And Measure String (C++) 370
  - Write String 256
- switching copy groups 171
- system requirements
  - AS/400 environment 395
  - MVS environment 375

## T

- tables
  - Begin Table 103
  - C sample 405
  - C++ sample 450
  - COBOL sample 462
  - End Table 148
  - example 37
  - examples 405
  - Put Text in Table Field 205
  - samples 405
  - steps to create 42
  - working with 36
- Tag
  - AFPDClass Class 317
  - AFPPage Class 325
- tags
  - group-level 12
  - page-level 12
  - Put Tag 201
  - Tag (C++) 317, 325
- text
  - AddText (C++) 327
  - C++ 326
  - IncludeText (C++) 324
  - orientation, illustration 331
  - Put Double-Byte Text 196
  - Put Text 203
  - Put Text in Table Field 205
  - Set Text Orientation 240
  - SetTextOrientation (C++) 330
  - Write String 256
- text strings, description 22
- text, GOCA
  - CharString (C++) 348
  - GOCA Character String 269
  - GOCA Set Character Parameters 287
  - SetCharSet (C++) 351
- TimeStamp
  - AFPDClass Class 317
- tools for AIX
  - AFPDMP 390
  - FLIPLIST 390
- Translate And Measure Double-Byte String C++
  - program call 368
- Translate And Measure Double-Byte String procedure
  - call 244
- Translate And Measure String
  - C++ program call 370

- Translate And Measure String *(continued)*
  - procedure call 246
- troubleshooting
  - AIX 393
  - MVS 387
  - OS/400 399
- types of data objects created by Toolbox 10

## U

- underscore
  - Begin Underscore 106, 149
- units
  - Set Units 242
- Unix System Services
  - environment variables 385
  - running MVS Toolbox on 385
- updating your font index
  - AIX 390
  - AS/400 397
- using fonts 46
- using Toolbox calls 53

## V

- validity
  - IsValid (C++) 318
- Vertical Move procedure call 249
- Vertical Move To 23
- Vertical Move To procedure call 251
- viewing online documentation (AIX) 389
- viewing the output file 47
- Vmove, PText Class 331
- VMoveTo, PText Class 332

## W

- working with code pages 44
- working with tables 36
- Write Double-Byte String procedure call 253
- Write String 23
- Write String procedure call 256

---

# Readers' Comments — We'd Like to Hear from You

AFP Toolbox for  
Multiple Operating Systems  
User's Guide  
Version 1 Release 1

Publication No. S544-5292-04

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

---

Name

---

Address

---

Company or Organization

---

Phone No.



Cut or Fold  
Along Line

Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE  
NECESSARY  
IF MAILED IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation  
Information Development  
Department H7FE, Building 004M  
Boulder, CO 80301-9817



Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold  
Along Line





Program Number: 5765-594, 5655-A25, 5798-AF2, and 5798-AF4



Printed in the United States of America  
on recycled paper containing 10%  
recovered post-consumer fiber.

S544-5292-04





Spine information:



AFP Toolbox User's Guide

Version 1  
Release 1